For Macintosh Programmers & Developers

MacTech™ MAGAZINE

Vol. 12, No. 9 • September 1996

Automated Quality Testing!

A True Crossplatform Library?

TOOLS OF THE TRADE
Exploring a crossplatform library

COMMUNICATION AND COLLABORATION
Extending Lotus Notes

QUALITY ASSURANCE
Cost effective testing

ADVANCED DEBUGGING
C++ Meets DebugStr

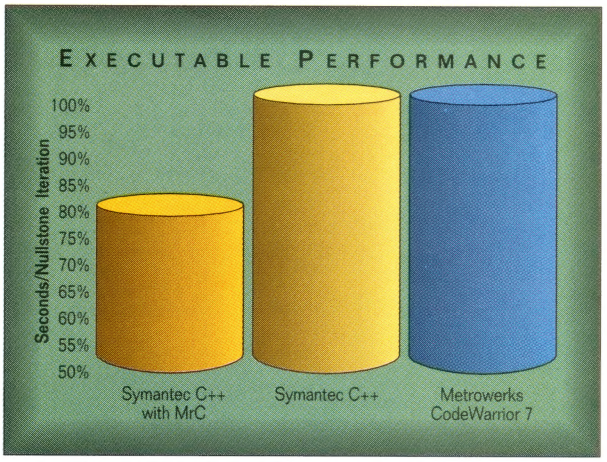
09
0 32128 74887 8

$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

Now, through a joint development agreement, Symantec and Apple™ Computer let you produce the fastest Power Mac code.

Symantec C++ for Power Macintosh™ now comes with MrC– Apple's new optimizing compiler.

Industry-standard Nullstone tests show that



*Code compiled using MrC runs an average of 22% faster than code compiled with the standard Symantec C++ compiler or Metrowerks Code Warrior 7™.**

size and color. Organizing and navigating a project has never been easier.

In addition, Symantec C++'s multi-threaded environment gives you the ability to edit and write code while you compile. And our visual architect lets you quickly draw the interface. Corresponding code is then generated automatically.

WHEN MRC GOES TO WORK, SYMANTEC C++ APPLICATIONS GET 22% FASTER.

applications compiled with MrC run an average of 22% faster.

DEVELOP FASTER APPLICATIONS FASTER.

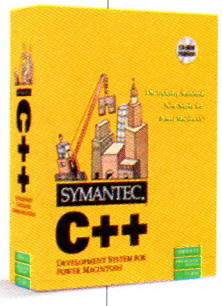
Not only can you develop the fastest Power Mac applications, you can write them fast, too. New AppleScript support lets you automate repetitive tasks. While the new linker provides fast turnaround for incremental builds.

With the fully integrated class browser, you'll quickly navigate your C++ class library. And support for templates and multiple inheritance further boosts your productivity.

ORGANIZE AND MANAGE PROJECTS EASILY.

The new Project Manager lets you organize and manage nested projects. You can display hierarchical groups as folders within a project window for easier organization.

The editor gives you unlimited split panes and full text formatting– font,



MRC COMPILER
produces the fastest Power Mac applications

APPLESCRIPT
automates the build process

NEW INCREMENTAL LINKER
provides fast incremental builds

TEMPLATE AND MULTIPLE INHERITANCE SUPPORT
increases productivity

NESTED PROJECTS AND FOLDERS
let you organize and navigate projects

MULTI-THREADED ENVIRONMENT
lets you edit and write code while compiling

VISUAL ARCHITECT
builds your interface visually

All in all, Symantec C++ is a great way to develop the fastest applications for Power Mac.

THREE CDS FOR THE PRICE OF ONE.

When you register as a Symantec C++ owner, you'll be enrolled in the Symantec C++ Subscription for Macintosh program. Subscribers will automatically receive two free product updates (on CD ROM) so you'll always have the latest features and tools.

Learn more about Symantec C++ on the Internet at www.symantec.com Or call 1-800-628-4777, Extension 9H25 for more information.

SYMANTEC.®

Eddy Award Winner** for **Best New Developer Tool
– MacUser Editors Choice Awards, 1993

"A distinct improvement over ResEdit."
– MacTech / MacTutor

"Resorcerer's data template system is amazing!"
– Bill Goodman, author of Compact Pro

"Nuke ResEdit! Resorcerer is mission-critical for us."
– Dave Winer, Userland Frontier

"The color pixel editors are wonderful! A work of art!"
– Dave Winzler, author of Microseeds Redux

"Every Macintosh developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

"...a wealth of time-saving tools."
MacUser Review, Dec. 1992

RESORCERER®

The Resource Editor for the Macintosh Wizard

Version 1.2.4

ORDERING INFO

Needs: ≥Mac Plus, ≥ Sys 4.2, 1MB
Likes: ≥Mac Plus, ≥ Sys 7.0, 2MB
32-bit clean, AU/X compatible

Price: $256 (decimal)
(Educational, quantity, or
other discounts available)

Includes: 500 page manual
60-day Money-Back Guarantee
Domestic UPS ground shipping

Payment: Check, PO's, or Visa/MC

Extras (call us):
COD, FedEx, UPS Blue/Red,
International Shipping

Downloadable Demos/Updaters:
AppleLink: Software Sampler
AOL: Software Libs/Development
CompuServe: MACDEV/Tools
or call us.

New 1.2 Features:

- **New 'cicn', 'ppat', 'crsr', 'acur', 'pltt', 'clut' editors**
- **Powerful icon family editing (all 9 icon types)**
- **Color pixel anti-aliasing, dithering, and lots more**
- **Complete 'PICT' disassembly and reassembly**
- **Resource sorting; ROM resource browsing**
- **120 template field parsing types now supported**
- **New insertion & deletion template field types**
- **Text-only 'PICT' resources**
- **Lots of improvements throughout**

- Easier, faster, more Mac-like, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Help, AppleEvent, OCE, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Well-designed, helpful developer tools being added all the time
- Relied on by thousands of Macintosh developers around the world

MATHEMÆSTHETICS, INC.
P.O. Box 298 • Boulder • CO • 80306-0298 • USA
Phone: (303) 440-0707 • Fax: (303) 440-0504
AppleLink/AmericaOnline: RESORCERER • Internet: resorcerer@aol.com

THE STAFF

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?** If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

DEPARTMENTS	Internet	CompuServe	AppleLink
Orders, Circulation, & Customer Service	cust_service@mactech.com	71333,1063	–
Press Releases	press_releases@mactech.com	–	–
Ad Sales	ad_sales@mactech.com	–	–
Editorial	editorial@mactech.com	71333,1065	–
Programmer's Challenge	prog_challenge@mactech.com	–	–
Online Support	online@mactech.com	–	–
Accounting	accounting@mactech.com	–	–
Marketing	marketing@mactech.com	–	–
General	info@mactech.com	71333,1064	MacTechMag
Online support area	http://www.mactech.com		

MACTECH MAGAZINE

MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology.

We are dedicated to the distribution of useful programming information without regard to Apple's developer status.

For information on submitting articles, ask us for our ***writer's kit*** *which includes the terms and conditions upon which we publish articles.*

Editors

Publisher • Neil Ticktin
Editor-in-Chief • Will Iverson
Editor • Matt Neuburg
Editor Emeritus • Scott T Boyd, QKS
Editor-at-Large • Eric Gundrum

Contributing and Technical Editors

Copland • Steve Kiene, Mindvision
Internet • Jon Wiederspan
MagicCap/TeleScript • Richard Clark, General Magic
OpenDoc • Tantek Çelik, 6prime corporation
Performance Programming • Jim Gochee, Connectix
Tips & Tidbits • Steve Sisak
Product Reviews • Ed Ringel
MacDev-1™ • Rich Siegel, Bare Bones Software, Inc.

Regular Columnists

Getting Started • Dave Mark, Metrowerks
Programmer's Challenge • Bob Boonstra
Insider's View • Jordan J. Mattson, Apple Computer, Inc.
Symantec Top 10 • Symantec Technical Support
URLs • Jim Straus, Key Internet Services

XPLAIN CORPORATION

Chief Executive Officer • Neil Ticktin
Chief Operating Officer • Andrea J. Sniderman
Controller • Brian Shin
Advertising Executive • Ruth Subrin
Senior Art Director • Judith Chaplin
Senior Copywriter/Designer • Andrea Luminati
Customer Service • Al Estrada
Vendor Relations • Annette Perez
Administrative Assistant • Susan Pomrantz
Network Administrator • Donal Corcoran
Software Engineer • Don Bresee

Board of Advisors • Steven Geller, Jordan Mattson, Blake Park, and Alan Carsrud

PRINTED WITH SOY INK™

This publication is printed on paper with recycled content.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

TABLE OF CONTENTS

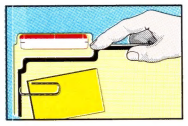
September 1996 • Volume 12, No.9

For Macintosh Programmers & Developers

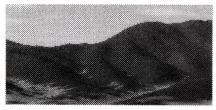
MacTech™ MAGAZINE

FEATURE ARTICLES

REGULAR COLUMNS



VIEWPOINT



By Will Iverson

This issue marks the first of my tenure as Editor-in-Chief of *MacTech Magazine*. Some of you may recognize me from my earlier role as Symantec's Macintosh Developer Relations Manager. I am excited to be at the helm of the magazine, and look forward to its growth over the next year. Write me at editor-in-chief@mactech.com with your ideas.

First, a brief comment about Apple Computer (leaving to others the lengthy non-technical dissertations on "what Apple should do"). Suffice it to say that the times may be a-changing, but that just makes for more excitement. Less than a year ago, Sun had a mixed future, and Java was barely on the horizon. Some were even wondering if the Power Macintosh might spell the end of Sun! Today Sun and Java are... Well, I think you get the point. The great thing about this industry is that both success and failure are fickle.

Overall, there's reason to be bullish. The Macintosh development tools market has never been stronger. There are more tools out there than ever before, and more people are learning to program. This is where the magazine comes in – we intend to expand coverage to meet the needs of this burgeoning market. One important piece of this is to challenge the ever-increasing marketing which inevitably comes with a healthy market. Who really is fastest, provides the best results?

Another component is the expansion of our Web site to form the new *MacTech NOW*. A problem with traditional print is the relatively long lag between timely news and the appearance of the information in print. We've been online for over a year, with over 1500 pages of back issues and online documentation. Now you can look to http://www.mactech.com/ for breaking news in the Macintosh Developer Tools community as well.

Mac OS 8, OpenDoc, SOM and Java

Pop quiz – what do Mac OS 8, OpenDoc, SOM and Java have in common? Give up? The futures of all four are tied together in a fundamental, albeit confusing, way.

A fundamental problem with the current implementation of object-oriented languages is the lack of a common calling mechanism – there is no easy way to mix and match pieces of object-oriented code. The most common place you run into this is trying to mix C++ libraries from two different compiler vendors in your project. For that matter, even C++ code compiled under different *versions* of the same compiler usually won't mix. This makes life for third-party library vendors very difficult, as they must maintain potentially dozens of different versions of their libraries. It also makes it very difficult for languages other than C++ (such as Smalltalk or Prograph) to make inroads into mainstream community.

There are two solutions. The easy but limited route is to develop a C++ **accepted binary interface**, or ABI. There are variants, but generally this model takes C++ as gospel for how object-orientation works and provides a relatively direct mapping. All compilers would then support this standard. This doesn't work across languages, and leads to annoyances like the `pascal` keyword in C. This is more or less the route Microsoft has taken with Windows.

The hard but robust solution is **SOM**. SOM (System Object Model) was developed by IBM, and in typical IBM fashion it is very robust, very powerful, and a royal pain to use.

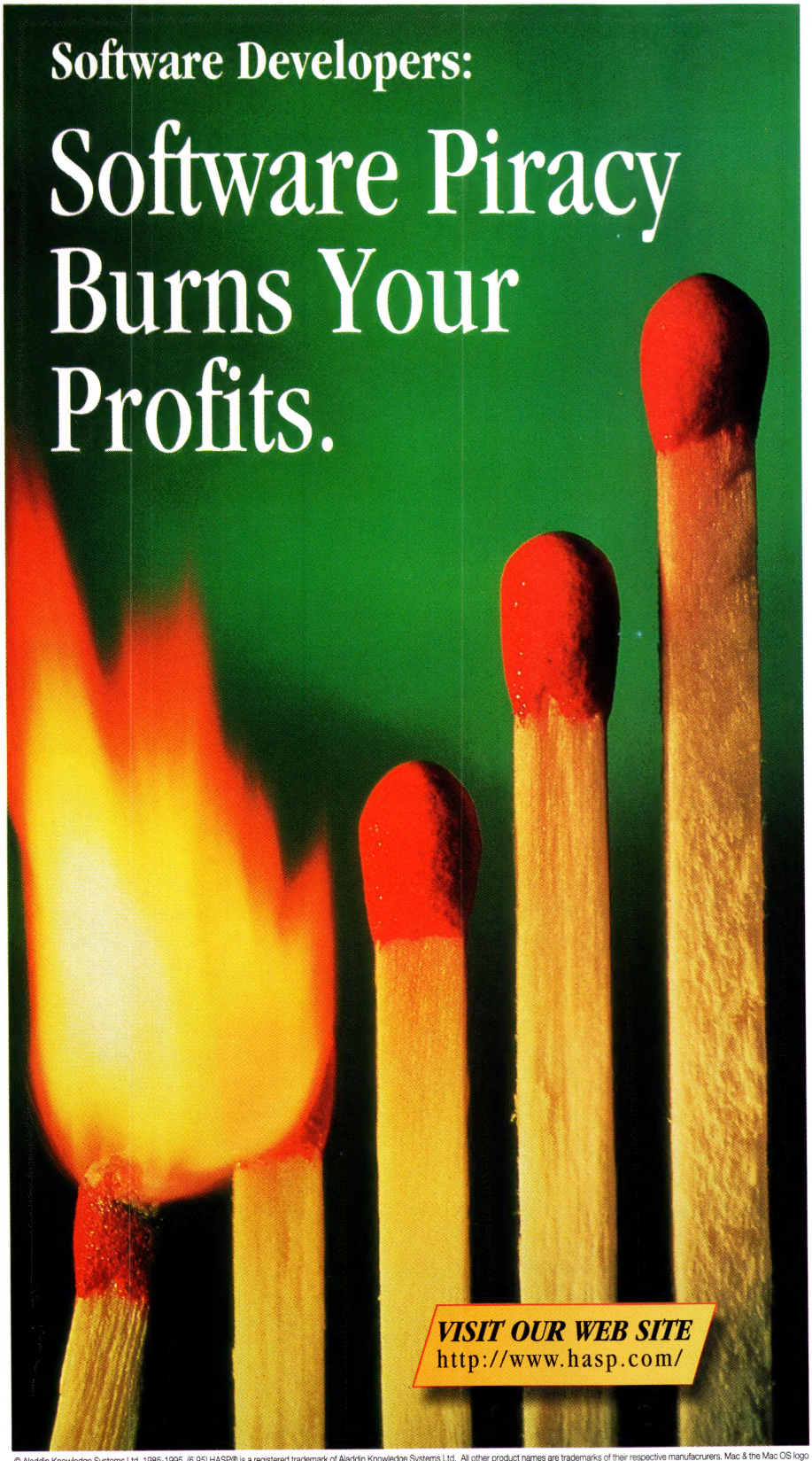
The Macintosh Way

The only company which can actually present and enforce such standards is Apple. Apple has decided on SOM as a standard, and we are already seeing this technology in OpenDoc. SOM will provide (among other things) the foundation on which the Mac OS 8 Appearance Manager is built. In addition, SOM libraries will finally allow us to share code between different tools.

SOM finally allows us to dispense with talking protocols, and to get on with generating content – in this case, code. There is a caveat, however. There are two ways to access SOM. The nasty way requires learning a new language and writing special .IDL header files, which are then mangled by an intermediate tool which actually generates the proper .h files. The easy way is something called **Direct to SOM**. (Do not confuse Direct to SOM with DSOM, which stands for Distributed SOM and is beyond the scope of this text.)

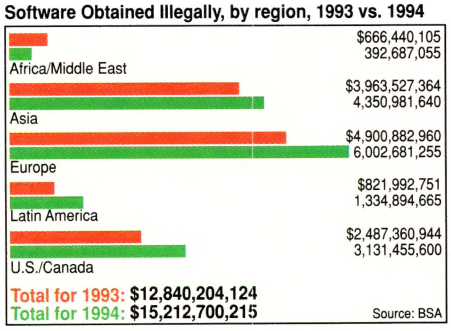
With Direct to SOM, you essentially keep working in C++ as normal, and things just work. The compiler is responsible for generating the proper code. This is also the problem with Direct to SOM – it basically involves rewriting the compiler. Metrowerks announced at MacHack that Direct To SOM was a principal component of CW10, and Apple has also announced Direct To SOM support in MrC. If your tool vendor doesn't support Direct to SOM, write and encourage them to do so.

So what does this have to do with Java and OpenDoc? OpenDoc suffers a critical flaw: a single misbehaved part can take down an entire document. Java, on the other hand, suffers from a lack of any structured, well-designed component architecture. At WWDC, Apple demonstrated a Java-based OpenDoc part which included access to native code and other neat things, like automatic persistence. The best part: it was roughly a page of code, a fraction of what it would have been in C++.



Each year, the illegal use of software consumes nearly 50% of your potential revenues. With the flames of piracy eating away at your profits, can you afford not to protect your software?

Software Obtained Illegally, by region, 1993 vs. 1994



MacHASP® is widely acclaimed as the world's most advanced software protection solution for Macintosh computers. Since 1984, thousands of leading Mac and PC developers have used over one million MacHASP and HASP keys to protect billions of dollars worth of software. Why? Because MacHASP's security, reliability, and ease-of-use led them to a simple conclusion: MacHASP is the most effective software protection system available.

Today, more developers are choosing MacHASP than any other software protection method. To learn why, and to see how easily you can increase your revenues, call now to order your MacHASP Developer's Kit.

1-800-223-4277

ALADDIN

The Professional's Choice

North America
Aladdin Software Security Inc.
Tel: (800) 223 4277, 212-564 5678
Fax: 212-564 3377
E-mail: sales@hasp.com
WWW: http://www.hasp.com/

Intl Office
Aladdin Knowledge Systems Ltd.
Tel: 972-3-537 5795, Fax: 972-3-537 5796
E-mail: aladdin@aladdin.co.il

United Kingdom
Aladdin Knowledge Systems UK Ltd.
Tel: 01753-622266, Fax: 01753-622262

France
Aladdin France SA
Tel: 1 40 85 98 85, Fax: 1 41 21 90 56

member of spa

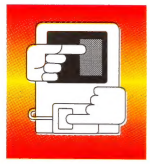
yes It runs with NetWare Developer tested only

Available and Ready for OS/2

MICROSOFT® WINDOWS™ COMPATIBLE

Mac™ OS

■ **Aladdin Benelux** 08894 19777 ■ **Aladdin Japan** 0426 60 7191 ■ **Aladdin Russia** 095 9230588 ■ **Australia** Conlab 3 8985685 ■ **Czech** Atlas 2 766085 ■ **Chile** Micrologica 2 222 1388 ■ **Denmark** Berendsen 39 577300 ■ **Egypt** Zeineldein 2 3604632 ■ **Finland** ID-Systems 0 870 3520 ■ **Germany** CSS 201 278804 ■ **Greece** Unibrain 1 6856320 ■ **India** Solutions 11 2218254 ■ **Italy** Partner Data 2 26147380 ■ **Korea** Dae-A 2 848 4481 ■ **Mexico** SiSoft 5 5439770 ■ **New Zealand** Training 4 5666014 ■ **Poland** Systherm 61 480273 ■ **Portugal** Futurmatica 1 4116269 ■ **Romania** Interactiv 64 153112 ■ **South Africa** D Le Roux 11 886 4704 ■ **Spain** PC Hardware 3 4493193 ■ **Switzerland** Opag 61 7169222 ■ **Taiwan** Teco 2 555 9676 ■ **Turkey** Mikrobeta 312 467 7504



By Dave Mark

The Peter Lewis Applet, After

Last month, we went through the first version of my color blinking text applet. As I told you then, I showed the applet to Peter Lewis (he of Anarchie and other cool Internet software fame) and he rewrote it. This month, we'll take a look at Peter's rewrite. As you read through Peter's code, bear this in mind. This applet definitely is not the only way to do things. Far more importantly, this version of CWBlink introduces a number of important Java topics, all of which you should learn about and eventually master. We'll aim to cover each topic in more detail in the coming months. For now, just enjoy.

The New CWBlink Project

The new version of CWBlink extends last month's example with a pushbutton that adds the ability to turn blinking on and off, and also offers a convenient, well-defined place to drop into your debugger. Figure 1 shows CWBlink with color blinking turned on.

The new CWBlink project is based on two Java source files instead of one. The first file, CWBlink.java, contains the heart of the applet, and is divided into two classes. The class `CWBlink` extends `java.applet.Applet`, and is the applet itself. The class `BlinkingText` extends the `Canvas` class (a generic drawing component class), implements the `Runnable` interface (which means it can be a thread – see last month's column), and defines a generic blinking text object. The `CWBlink` class creates and starts the `BlinkingText` object.

The second file, AppletFrame.java, is needed only if you want your applet to run standalone, replacing the applet frame with your own frame. This source file is definitely worth understanding. We'll start off by listing both source code files (along with the associated HTML file), then cover the highlights of each file.

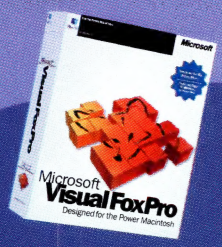
Figure 1. The CWBlink applet with color blinking turned on

By the time you read this, I should have my own Web site, at http://www.spiderworks.com/dmark/. This site is maintained by my brother, Stu (thanks, Stu!), and is intended as a public resource. It will include links to Getting Started code kept on the MacTech Web site. If you have a cool programming-related site and you'd like to cross-link with my Web page, send email to stumark@spiderworks.com and he'll set it up. If you have any comments (good or bad) about the site, especially with things you'd like to see on the site, send your comments to Stu as well.

If you like the experience of creating your project from scratch (as I do), launch CodeWarrior, create a new project using the Java Applet stationery, then create three new source code files. Here's the source code for CWBlink.java:

```
package com.metrowerks.example.CWBlink;

import java.awt.*;

public class CWBlink extends java.applet.Applet
{
  private Button blinkButton;
  private BlinkingText blinkText;

  public String getParameter(String name)
  {
```

Ever since Fred got the new Visual FoxPro for Power Macintosh, he catches the Frisbee every time.



And perhaps more importantly, he also gets his work done in record time. Thanks to Microsoft® Visual FoxPro™ 3.0 for Power Macintosh,® one might say that Fred's productivity is positively soaring.

After all, the powerful object technology of Visual FoxPro lets him create class libraries with inheritance and encapsulation and therefore reuse code quickly and easily. Fred has no trouble integrating data, either, whether it's located in applications, client/server databases, or local datastores. And because of Visual FoxPro's complete binary compatibility, he can develop in the familiar Mac™ environment and rest assured that all code, including Visual FoxPro objects, will run in Windows® 3.1 and Windows 95 operating systems with no modifications.

To find out more about the power of the Visual FoxPro 3.0 database management system, just point your mouse to http://www.microsoft.com/devonly/. Or give us a call at **(800) 621-7930, Dept. A334DS**, in the 50 United States.*

Microsoft
WHERE DO YOU WANT TO GO TODAY?™

*In Canada call (800) 563-9048; outside the U.S. contact your local subsidiary.

```
    String result = null;

    try
    {
      result = super.getParameter(name);
    }
    catch ( Exception e )
    {
      result = null;
    }

    return result;
  }

  public void init()
  {
    Panel tempPanel;

    String att = getParameter("speed");
    int speed = (att == null) ?
      500 : (1000 / Integer.valueOf(att).intValue());

    setLayout( new BorderLayout() );

    att = getParameter("blinker");
    String blinkString = (att == null) ?
      "CodeWarrior!!!" : att;

    blinkButton = new Button( "Blink" );

    tempPanel = new Panel();
    tempPanel.add( blinkButton );
    this.add( "North", tempPanel );

    blinkText = new BlinkingText( blinkString, speed );

    tempPanel = new Panel();
    tempPanel.add( blinkText );
    this.add( "Center", tempPanel );

    resize( 570, 170 );
  }

  public void start()
  {
    blinkText.start();
  }

  public void stop()
  {
    blinkText.stop();
  }

  public boolean action(Event evt, Object arg)
  {
    if ( "Blink".equals(arg) )
    {
      blinkText.ToggleBlinking();
    }
    return true;
  }

  public static void main(String args[])
  {
    com.metrowerks.AppletFrame.startApplet(
        "com.metrowerks.example.CWBlink.CWBlink",
        "Blink", args);
  }
}


class BlinkingText extends Canvas implements Runnable
{
  private Thread blinkThread = null;
  private String blinkString;
  private Font font;
  private int speed;
  private boolean isBlinking = false;
  private Color[] letters;
  private boolean[] is_black;
  private int current_letter = 0;
  private int old_width, old_height;

  private Color RandomColor()
  {
    int red, green, blue;

    do {
      red = (int)(Math.random() * 256);
    } while ( red > 0x8000 );

    do {
      green = (int)(Math.random() * 256);
    } while ( green > 0x8000 );

    do {
      blue = (int)(Math.random() * 256);
    } while ( blue > 0x8000 );

    return new java.awt.Color( red, green, blue );
  }

  public BlinkingText( String blinkString, int speed )
  {
    this.blinkString = blinkString;
    this.speed = speed / blinkString.length();
    this.font = new java.awt.Font( "TimesRoman",
                                   Font.PLAIN, 64 );
    this.letters = new Color[blinkString.length()];
    this.is_black = new boolean[blinkString.length()];

    for ( int i = 0; i < letters.length; i++ )
    {
      letters[i] = RandomColor();
      is_black[i] = true;
    }
    resize( 530, 70 );
    repaint();
  }

  public synchronized void ToggleBlinking()
  {
    isBlinking = ! isBlinking;
    notify();
  }

  public synchronized void PaintLetter( int the_letter )
  {
    Rectangle b = bounds();
    int width = b.width;
    int height = b.height;
    Graphics g = getGraphics();

    if ( old_width == width && old_height == height )
    {
      int x = 0;
      int y = height;

      g.setFont(font);
      FontMetrics fm = g.getFontMetrics();

      for (int index=0; index<the_letter; index++ )
      {
        int w = fm.charWidth(blinkString.charAt(index));
        x += w;
      }

      int w = fm.charWidth(blinkString.charAt(the_letter));
      g.setColor( isBlinking ?
                  letters[the_letter] : Color.black );
      g.clearRect( x, 0, w, height );
      g.drawString( blinkString.substring(
                  the_letter,the_letter+1), x, y );

      is_black[the_letter] = !isBlinking;
    }
    else
    {
      paint( g );
    }
  }
```

Announcing

MICROGUARD PLUS.™

Why so many developers are switching to MicroGuard copy protection

MicroGuard is committed to uncompromising technological superiority

"Technology at its peak" is our commitment to you. That is why we have brought you **MicroGuard Plus**™. And, MicroGuard Plus is **100% backwards** compatible with MicroGuard. This means MicroGuard and MicroGuard Plus can be used interchangeably. Just as we promised!

MicroGuard has now surpassed its own technological lead, actually improving on the best

MicroGuard Plus is everything MicroGuard is, plus 40-bit encryption, two additional passwords, 64-Bit Array, 32-byte public area, 45% size reduction, enhanced counter and more. In addition, MicroGuard Plus offers two new utilities: **QuickGuard**™ and EasyGuard™ allow you to protect your applications without touching your source code. The only feature that is not plus is the price. :-)

MicroGuard offers you the most sophisticated network protection

Our network protection, **MicroGuard Net**™, is so superior, we had to hire an Apple network engineer to execute our specifications.

MicroGuard is the best selling Macintosh key in the world

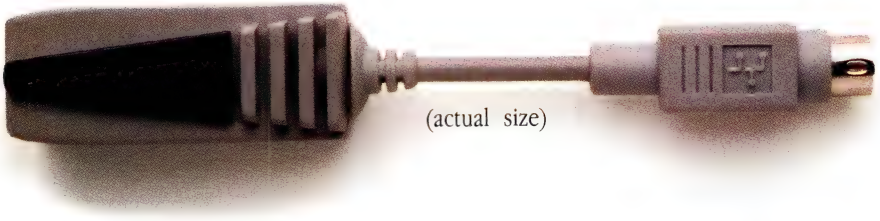
MicroGuard sells more Macintosh copy-protection keys than anyone else in the world!

MicroGuard is the only key developed by Mac developers, and is the first and only 100% ADB savvy key

We have been developing Mac applications as a seed development house since 1984. We are not a PC protection company that has come to you with a Mac product. MicroGuard is fully ADB savvy and offers extended addressing. Unlike other protection devices, MicroGuard never clashes with other keys. Only MicroGuard offers this level of sophistication.

MicroGuard delivers developer support within 24 hours

We will answer any inquiry you have within 24-hours. We also have a fully loaded AppleLink bulletin board which contains all our libraries, tech notes, Q&A and nearly everything you'll ever need!



(actual size)

For more information and to order a Developer's Kit or to receive a free CD ROM about MicroGuard, please contact us at:

MicroGuard USA:	Tel: (303) 320-1628	• Fax: (303) 320-1599	• AppleLink: M.GUARD
International:	Tel: (972) 3 558-2345	• Fax: (972) 3 558-2344	• AppleLink: MICROGUARD

Mac™OS

Pirate Busters

```
public synchronized void paint(Graphics g)
{
  Rectangle b = bounds();
  int width = b.width;
  int height = b.height;
  old_width = width;
  old_height = height;
  int x = 0;
  int y = height;

  g.setColor(Color.black);
  g.setFont(font);
  FontMetrics fm = g.getFontMetrics();

  g.clearRect( 0, 0, width, height );
  if ( isBlinking )
  {
    for (int index=0; index<blinkString.length();
        index++ )
    {
      int w = fm.charWidth(blinkString.charAt(index));

      g.setColor( letters[index] );

      g.drawString(
        blinkString.substring(index,index+1), x, y);
      x += w;
    }
  }
  else
  {
    g.drawString(blinkString, x, y );
  }

  for ( int index=0; index<blinkString.length(); index++ )
  {
    is_black[index] = !isBlinking;
  }
}

public void start()
{
  stop();
  blinkThread = new Thread(this);
  blinkThread.start();
}

public void stop()
{
  if ( blinkThread != null )
  {
    blinkThread.stop();
    blinkThread = null;
  }
}

public void run()
{
  while (true)
  {
    synchronized ( this )
    {
      try
      {
        wait( 1 );
      }
      catch (Exception e)
      {
      }

      if ( isBlinking || !is_black[current_letter] )
      {
        if ( isBlinking )
        {
          letters[current_letter] = RandomColor();
        }
        PaintLetter( current_letter );
      }
      current_letter++;
      if ( current_letter >= blinkString.length() )
      {
        current_letter = 0;
      }
    }
  }
}

public void finalize()
{
  stop();
}
}
```

Save the source code as **CWBlink.java** and add it to the project. Next up, here's the code for **AppleFrame.java**:

```
package com.metrowerks;

import java.awt.*;
import java.applet.Applet;

public class AppletFrame extends Frame
{
  public static void startApplet( String className,
          String title, String args[])
  {
    Applet a;
    Dimension appletSize;

    try {
      a = (Applet)
          Class.forName(className).newInstance();
    } catch (ClassNotFoundException e) {
      return;
    } catch (InstantiationException e) {
      return;
    } catch (IllegalAccessException e) {
      return;
    }

    a.init();
    a.start();

    AppletFrame f = new AppletFrame(title);

    f.add("Center", a);

    appletSize =  a.size();
    f.pack();
    f.resize(appletSize);
    f.show();

  }

  public AppletFrame(String name)
  {
    super(name);
  }

  public boolean handleEvent(Event e)
  {
    if (e.id == Event.WINDOW_DESTROY)
    {
      dispose();
      return true;
    }

    return super.handleEvent(e);
  }
}
```

Save this code as **AppleFrame.java** and add it to the project as well. OK, last file. It's the HTML file:

```
<title>Blinking CodeWarrior</title>
<hr>
<applet codebase=CWBlink
    code="com/metrowerks/example/CWBlink/CWBlink.class"
    width=530
    height=120>
```

Continued on page 14

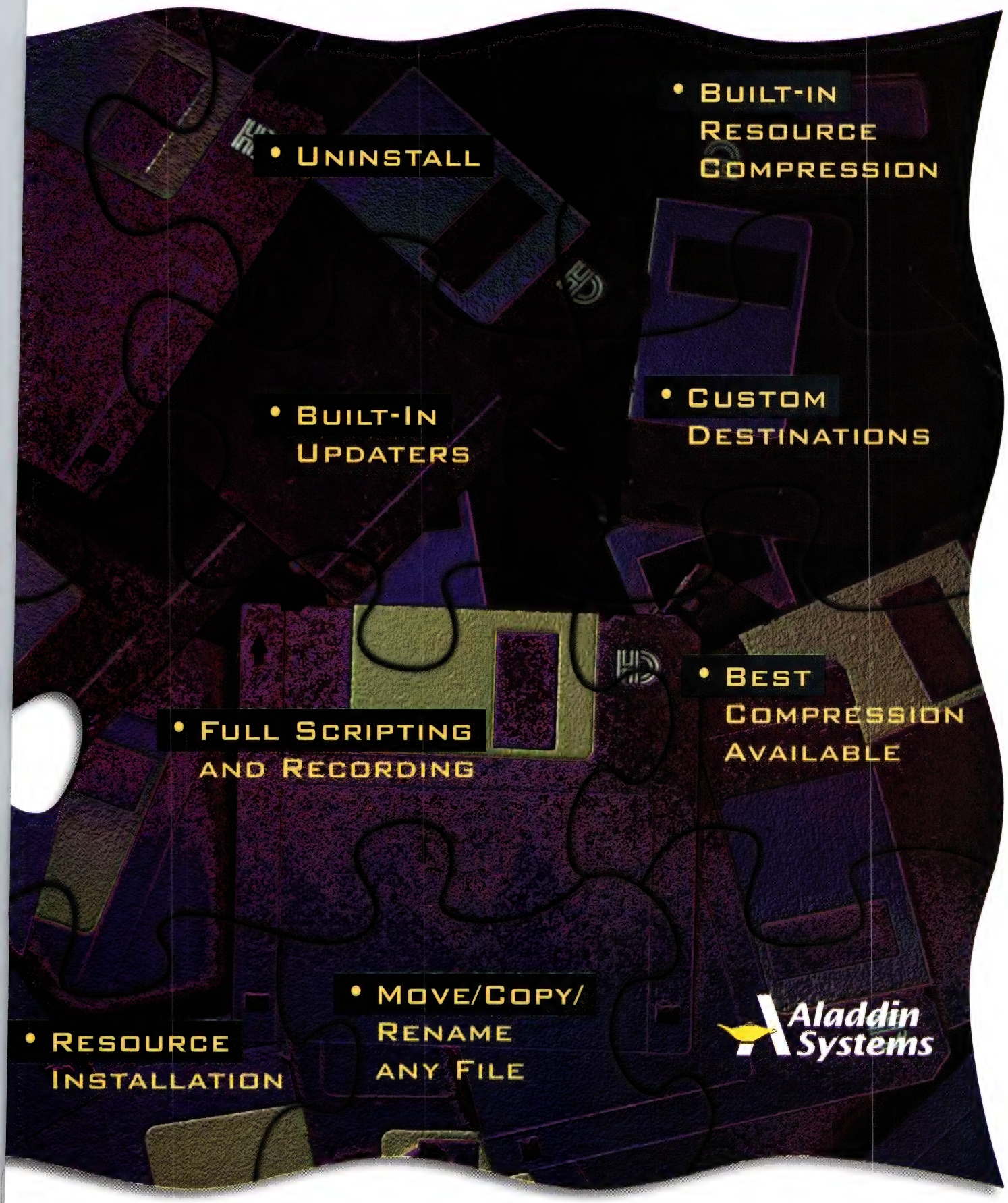
Our competitors' habit of charging for each piece of their installer program is a bit puzzling.

- CREATE UP TO 128 EXPERT-INSTALL "PACKAGES"
- IMPROVED DEVELOPER CUSTOMIZATION

The reason we call StuffIt InstallerMaker 3.0 the Complete Installation Solution is because we include **everything you need** to prepare a professional package of files for installation on your user's machines. An installer, updater, and an uninstaller are just three components included for the same price one of our fine competitors charges for just the installer. Using our installer reduces technical support calls due to end-user errors during installation, saves disks (if distributing on floppies) or download time (if distributing on-line). You will save more than the cost of the Installer license by significantly reducing distribution costs. And the puzzle will be solved. You'll have every piece in place.

Download a **free,** fully-functional copy of StuffIt InstallerMaker 3.0 from www.aladdinsys.com, or call (408) 761-6200 and ask for Developer Sales.

STUFFIT INSTALLERMAKER 3.0


• Uninstall
• Built-in Resource Compression
• Built-In Updaters
• Custom Destinations
• Full Scripting and Recording
• Best Compression Available
• Move/Copy/ Rename any File
• Resource Installation
Aladdin Systems

Continued from page 10

```
  <param name=blinker value="CodeWarrior!!!">
  <param name=speed value=2>
</applet>
<hr>
<a href="CWBlink.java">The source.</a>
```

Save this one as CWBlink.html and add it to the project as well. (You don't have to add the HTML file to the project, but it does make editing the file more convenient – you just double-click on the file name in the project window.)

Once your source code and HTML files are entered and added to the project, go to the **Preferences** panel, **Java Project** pane, select **Class Folder** from the **Project Type** popup, and enter **CWBlink** in the **Folder Name** field. This will store any class files generated by this project in a folder hierarchy inside a folder named CWBlink. This option works well if you plan on serving your applet on a Web page.

Select **Make** from the **Project** menu. CodeWarrior will compile your Java code and generate the appropriate .class files. To run your applet, drag your HTML file onto the Metrowerks Java application. If you want to use the debugger, be sure the debugger is enabled and that you have the MetroNub extension installed, then drag the .class files you want to debug onto the MW Debug application. When the debugger launches, click on your source code file name in the debugger window. If your source code file is not in the same folder as the .class file, you'll be prompted to locate the source file. This can be a hassle, so to prevent it from happening you might want to keep your source files in the same folder as your .class files, even though this is a slight pain to set up.

Once the debugger finds your source code, it lets you do all the normal pre-running things like setting breakpoints. Note that the debugger window for one class will let you set breakpoints only in the functions that belong to that class.

Try this one: Drag CWBlink.class onto the MW Debug application. In the debug window that appears, click on CWBlink.java. When the source code appears, scroll down to this function:

```
public boolean action(Event evt, Object arg)
{
  if ( "Blink".equals(arg) )
  {
    blinkText.ToggleBlinking();
  }
  return true;
}
```

Set a breakpoint next to the call to `blinkText.ToggleBlinking()`. Now go back to the Finder and drag the HTML file onto the Metrowerks Java application. When the applet starts running, click on the **Blink** button. If all was set up properly, you'll pop into the debugger at the breakpoint.

Source Code Highlights

Normally, this is where I'd step through every single line of code in the applet. But since you've seen a lot of this code before (last month's column), I'll just run through the highlights. If you don't get something about the code, don't worry. The important thing to take away from this month's column is list of concepts to read up on.

Take a look at the beginning of CWBlink.java:

```
package com.metrowerks.example.CWBlink;

import java.awt.*;
```

The first thing you notice is the `package` statement. Packages are like C and C++ libraries. The `package` statement at the beginning of a Java source file gives the collection of classes in this file a name. In this case, the `CWBlink` and `BlinkingText` classes are grouped in the package `com.metrowerks.example.CWBlink`. It's important that you place the .class files generated from this source file into a directory structure that matches the package name. In this case, the two files CWBlink.class and BlinkingText.class must be placed in a folder with the path com:metrowerks:example:CWBlink. Fortunately, CodeWarrior does this for you automatically.

You've already seen how to access a package: use the `import` command. In this file, we import the `java.awt` package.

Note that the `getParameter()` code was pulled into its own method. Smart move. Gee, why didn't I think of that?

The `init()` method has some important new stuff in it. First, notice the use of the `Panel` class. A Panel is sort of like a PowerPlant View. Java uses the concept of containers and components. A container is just like it sounds, a containing view. A component is an interface element, like a pushbutton or checkbox. A Frame is an applet's outermost container. A Panel might contain components and other containing classes.

To simplify things, think of a Frame as a Window, and a Panel as a collection of elements you want to group together. If your applet runs in a Web browser, the outer Frame is created for you. If you plan to run the applet standalone, you'll need to create the Frame yourself. More on that in a minute (when we discuss AppletFrame.java).

The `init()` method loads the `speed` parameter (see last month's column), then sets a layout for the current Frame. You should read up on layouts (any Java book worth its salt will cover them), but here's the basics.

Every container has a layout that defines how contained elements are to be arranged. For example, the `BorderLayout` (see java.awt.BorderLayout.html) lays out a container using members named `"North"`, `"South"`, `"East"`, `"West"` and `"Center"`. A component added as `"North"` gets placed at the top of the container. A component added as `"Center"` gets the space left over when the other components are laid out.

Once the `init()` method sets the layout, it gets the blinking text parameter, then creates a pushbutton with the text `"Blink"`. The button is added to a panel and the panel added to the current container. Since `"North"` is used, the button will appear at the top of the container.

Next, a new `BlinkingText` object is created and added to the center of the container. Finally, the container is resized. Play with this stuff. Try using `"South"` or `"East"`. Change the parameters passed to `resize`.

Another important CWBlink method is `main()`. You can actually delete this method, and remove the AppletFrame.java file from your project, if you only want to run the applet from a Web page (from within a pre-built applet Frame). If you want your applet to run standalone, though, you'll need to build an applet Frame yourself. AppletFrame.java does this. When you are running standalone, `main()` will get called automatically, and the `AppletFrame.startApplet()` method will get called.

To learn how all this works, read through the source in AppletFrame.java. It's not very long, and will serve as a nice introduction to Frames. Even better, you can just add the AppletFrame.java file to your own applet projects.

Finally, the `BlinkingText` class demonstrates the extremely important topic of threads. As you know, Java is multi-threaded. For example, there is a garbage-collecting thread that ensures that an object that is no longer referenced is deallocated, so that your Java heap won't end up hopelessly fragmented. The `Thread` class is very important. Read up on it (check out java.lang.Thread.html). Think of a Thread as a separate little flow of control within your main process.

There are several ways to create a Thread. You can subclass `Thread` and override the necessary methods (such as `run()`). Alternatively, you can implement the `Runnable` interface, which is what `BlinkingText` does. Once you start a thread, the VM keeps running your threads until they all die.

The `synchronized` keyword marks a block of code so that only one thread can access it at a time. This means that if one thread starts executing a synchronized method, any other thread that wants to run the same method has to wait until the first method is finished. This keeps two threads from confusing each other – by messing with the same variables, for example.

Till Next Month

We intend to dig into all of these topics in detail in future columns. In the meantime, spend some time with the Java HTML docs and read a good Java book.

By Jeremy Vineyard

Software Testing With Virtual User

A cost-effective proving ground for software

Introduction

It is continually frustrating to have new applications crash soon after being installed. Such products give the impression that insufficient work was done before shipping to ensure their quality.

In software companies today, it is standard procedure to test a product to ensure its quality and reliablity before shipping. Quality Assurance (QA) is a common name for such testing. QA is becoming increasingly important as applications take on ever-increasing size and complexity. There can be many thousands of variations in the ways a user might interact with a software product, and a QA tester must check these to verify that everything works correctly. So QA testers need the tools to get their job done effectively.

There are numerous ways to improve the software testing process, but one of the most effective has been the automated testing tool, whereby QA testers can set up specific tests and suites of tests to be run by the computer. This frees them from many hours of drudgery, thus saving both time and money by allowing them to refocus their energies on tasks that need more human interaction, spreading the range of the testing eventually completed. This also leads to more reliability in the software.

You don't have to have a dedicated QA department to be able to test your products. Even if you are a small developer, it is a good idea always to verify the quality of your products before they are shipped, and automated tools can help. One of the most popular automated testing tools for the Mac is **Virtual User**.

What is Virtual User?

Virtual User (VU) is an automated testing tool that allows a computer to emulate a human user, performing actions such as clicking the mouse and typing keys. This computer acts as a **host**, and controls other computers just as a person would. One or more **target** computers act as agents receiving instructions from the host.

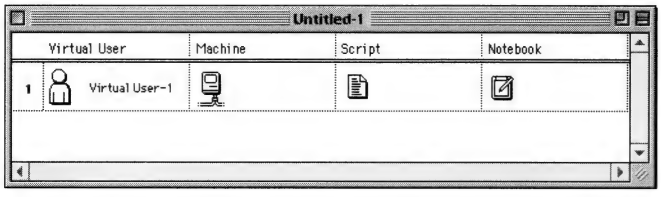


Figure 1. Virtual User

The VU environment consists of an application that compiles and run **scripts**. VU scripts must be edited separately in a text editor, such as MPW or BBEdit.

VU links computers together over an AppleTalk network. The minimum setup for a VU testing system is the VU software package and two computers (one host and one target).

VU is developed by Apple Computer, Inc., and is available for under $100 through most Macintosh developer catalogs. [*It comes for free if you're already subscribing to the ETO CD. Apple lists the price as $150 on their Web site, at* http://dev.info.apple.com/TPC/VU.Datasheet.html, *but says $79 in the printed APDA catalogue. Go figure. – man*]

Jeremy Vineyard works for Farallon Computing, Inc., and is in charge of automated software testing for the Macintosh. His interests outside of software testing include developing games, software utilities, and application class libraries. He can be reached at jeremyv@farallon.com.

neoAccess™

Cross-Platform Object Database Engine

Power Too Abundant to Meter

Powerful

NeoAccess and NeoShare are the most powerful object-oriented database engines available. They display electrifying performance—up to ten times that of competitors. Behind an elegant programming interface is a high performance query engine utilizing: extended binary trees and binary search algorithms tuned for short access times, dynamically combined, collapsed, and compressed indices, and object caching for lightning fast access to previously used objects.

No Runtime Fees

Get the power of NeoAccess and avoid the expense and administrative hassle of feeding the runtime fees meter. You pay one affordable price no matter how many copies of your application you sell or use.

Cross Platform

Others may promise cross-platform development tools—NeoLogic delivers. NeoAccess and NeoShare consists of C++ classes designed for use with popular compilers and application frameworks on Windows® (3.x, '95 and NT), Macintosh® (M68k and PPC), and Unix™ (all flavors) platforms. Full source code is available so both products can even be used with custom frameworks.

New in NeoShare Version 2.0

Cross-platform and internet-savvy NeoShare 2.0 is now shipping! NeoShare extends standard application frameworks including; Metrowerks' PowerPlant, Symantec's THINK Class Library and Apple's MacApp on the Macintosh, and Microsoft's Foundation Classes and Borland's OWL in Intel-based environments.

neoShare™

Client/Server Object Database Engine

Scalable

NeoShare extends the core features of NeoAccess to client/server applications. NeoShare includes the NeoAccess Toolkit and everything you need to create data intensive distributed applications. Its advanced client/server architecture provides shared access to objects by multiple inter- and intra-networked clients. Combine client and server functions into a single application or create separate client and server applications. With NeoShare you have complete flexibility in the design of your distributed systems.

Proven

Thousands of commercial and in-house developers have already found that NeoLogic's technology enabled them to build fast, lightweight and powerful applications in record time. That's why NeoAccess and NeoShare based applications are already operating on millions of computers. Tap into the power for your next development project!

neo•logic™

Powering Development of Object-Oriented Applications

NeoLogic Systems 1450 Fourth St., Suite 12 v. 510.524.5897
neologic@neologic.com Berkeley, CA 94710 f. 510.524.4501

Limitations Of Virtual User

There are many things that an automated tool cannot do, and it is important to realize these limitations before planning your automated test suites. An automated tool cannot tell when something "looks right" on the screen. VU can't tell you when an icon or window is pretty or ugly or misaligned.

VU is fairly unintelligent. You can tell it to "Move window 'My Window' to (x,y)", but you can't tell it to "Open icon for hard drive in Finder." Also, if the target machine crashes, VU doesn't know about it and will keep trying to run the script. The script can be paused and the machine restarted to allow the script to resume. Proper debugging techniques are essential for determining the events that led up to the crash. Some of these techniques will be explained later in the article.

Advantages of Virtual User

VU is most effective at highly repetitive tests that may have many variations. One example might be to select every control in every dialog in the application, making sure that each click produces the appropriate dialog, action, etc. This would be a very tedious task for a human to accomplish, but the computer doesn't care about tediousness, making it the ideal tester for the project.

Another use of VU is to write a test that can be run after every internal build of a product (development, alpha, beta, final candidate) to verify that nothing was broken by the code changes made to the previous version. VU can also be used to set up automatic bug verification, by acting out the steps necessary to reproduce a bug. With this capability, the automatic bug verification can be run on every new release of a product to ensure that the bug is fixed and that it doesn't sneak back into the code.

Understanding Virtual User Scripts

Each automated script has an entry point specified by the `script` statement in VU:

```
script TestControlsAndWindows()
begin
  # Do automated tests here.
end;
```

From then on, the VU script is much like the actions of a person. You tell it to click on windows or buttons, move the mouse, type keys, and more. Here is an example of a script that will test some menu items:

```
script TestMenuItems()
begin
  # Select the menu item "Show Script Window" from the "Windows" menu.
  select [menuItem title:"Show Script Window"
          menu:[menu title:"Windows"]];

  # Wait 10 seconds for the window to appear.
  wait(10);

  # Verify that the "Script Window" window appears.
  if match [window ordinality:1 title:"Script Window"]
    println("The window 'Script Window'
             opened correctly.");
  else
    println("ERROR! The window 'Script Window'
             did not open.");
end;
```

In addition to scripts, VU supports functions that act as extensions to the script. These are called **tasks**. A task is a procedure with a list of parameters and an optional return value.

```
task DoTheRightThing(var1 := "default value", var2)
begin
  # Do the right thing here.

  # Return a value (optional).
  return "result";
end;
```

VU collects information about the target computer's environment using (as we have already seen) the `match` statement. The `match` statement looks for a specific environment element, using descriptor traits such as the element's title or ordinality.

VU can "see" any menu or window and any control that is implemented with the Control Manager and stored in the window's list of controls. However, because the List Manager doesn't provide a standard API for accessing the contents of a list, VU cannot see a list or the items inside it. VU can also see dialog items such as user items, static text, edit text, icons, and pictures.

```
# Find the menu called "Testing".
theMenu := match [menu title:"Testing"];

# Find the frontmost window.
theWindow := match [window ordinality:1];
```

The `collect` statement is similar to `match` in that it collects all the elements of a certain type into a list.

```
# Keep a list of all the open windows.
windowList := collect [window];
```

VU then interacts with the environment, using such keywords as `select`, `drag`, `close`, `type`, and `click`. VU accesses common Macintosh objects such as windows, buttons, scroll bars, and menus.

```
# Select the menu item.
select [menuItem title:"Show Clipboard"
        menu:[menu title:"Edit"]];

# Move the window.
drag [window title:"Clipboard"] relative:{30, 30};

# Type characters into the window.
type keystrokes:{"This is some text"};

# Select a button.
select [button title:"OK"];

# Close the window.
close [window ordinality:1 title:"Clipboard"]!;
```

Tip: If possible, implement your application windows as modeless dialogs. VU recognizes the user item element in a dialog, allowing user items to indicate to VU where non-standard user interface elements are located.

Debugging Virtual User Scripts

VU provides a log file feature, by which information from within the script can be written out, providing the scripter with

information about the current run-time state of the script. One of the most effective ways to debug an automated script is to log with the `println` statement anywhere anything important happens or changes. (The `{}` syntax substitutes the actual values of the variables for the variable names in the string.)

```
task MyTask(var1, var2, var3)
begin
  println("MyTask({var1}, {var2}, {var3})");

  println("Starting batch processing.");
  StartBatchProcessing();
end;
```

Tip: Because there is no type checking in VU, it is a good idea to log the parameters that are input into every task to make sure that the correct values were passed.

Extending Virtual User with Libraries

One of the most useful features of VU is the ability to split commonly used tasks into reusable files called **libraries**. To use all of the tasks in a library, the `Libraries` statement is used as shown:

```
# This statement makes all the tasks in the file called "Special Tools.vulib"
# accessible to this script.
Libraries "Special Tools.vulib";
```

Libraries are commonly used to group tasks with common actions into smaller and more manageable files, such as Finder Tools.vulib, My App Tools.vulib, etc.

Tip: Because large projects may have dozens of library files, it is a good idea to establish a naming convention for both the names of the library files and the names of the tasks within the files. One might, for instance, append "Tools.vulib" to the name of every library file, and use a unique two-letter prefix for every task in the library.

```
# This task is in the library "Finder Tools.vulib".
task FT_EjectDisk()
begin
  # Eject the disk with a command-key combination.
  pressKey keystrokes:{commandKey};
  type keystrokes:{'E'};
  releaseKey keystrokes:{commandKey};
end;
```

Extending Virtual User with Globals

Another useful feature of VU is support for global variables. To declare a global variable, simply place the `global` keyword before the variable name when you define it.

```
# Once this variable is defined, it can be accessed from anywhere.
global gMyGlobal := 1000;
```

To access a previously defined global variable, again attach the `global` keyword before using the variable.

Here is an example of how using global variables can save considerable time and effort. This script:

```
task MA_DoThis(appVersion := '1.0', var1, var2, var3)
begin
end;

task MA_DoThat(appVersion := '1.0', var1, var2)
begin
end;

script MyScript()
begin
  MA_DoThis('1.0', 10, 20, 30);
  MA_DoThat('1.0', "test", "do");
end;
```

can be simplified to this:

```
task MA_DoThis(var1, var2, var3)
begin
  # Now the variable 'gAppVersion' is using the global value.
  global gAppVersion;
end;

task MA_DoThat(var1, var2)
begin
  # Now the variable 'gAppVersion' is using the global value.
  global gAppVersion;
end;

script MyScript()
begin
  global gAppVersion := '1.0';

  MA_DoThis(10, 20, 30);
  MA_DoThat("test", "do");
end;
```

The advantages may seem small in this simple example, but once you start writing scripts with hundreds or even thousands of separate tasks being called, it can be difficult to pass variables several levels deep through the chain. Using globals allows variables to be accessed from anywhere within the script.

Tip: For applications whose user interface is changing rapidly, insteading of searching all of your scripts and libraries every time the text for a menu item, window, or button is changed, use a global variable that is declared at the beginning of every script, to hold the current state of the interface item.

```
# This task must be called before the globals can be used.
task MA_DeclareGlobals()
begin
  global toolsMenu := [menu title:"Tools"];

  # If the name of the paint tool menu item ever changes, we have only to
  # change it in one place, and all other scripts will be using the correct
  # value. This keeps us from having to replace the string everywhere it
  # occurs in all of the scripts.
  global paintToolMenuItem :=
          [menuItem title:"Painter" menu:toolsMenu];
end;

task MA_SwitchToPaintTool()
begin
  select global paintToolMenuItem;
end;
```

Extending Virtual User with External Tools

If you find that you have reached the limitations of the VU language, it is possible to write an external tool for VU. An external tool is an application that communicates with VU by sending and receiving Apple events. Because the external tool can be written in a more powerful language such as C/C++ or Pascal, the VU language can be extended. Templates and examples for creating external tools come with VU, eliminating much of the work.

Additional Features of Virtual User

VU has many of the features of a modern programming language, including if-else statements, for/while loops, list- and string-processing operators, and more.

VU has a built-in regular expression matching system that allows you to write scripts that match environment elements based on regular expressions, rather than simple strings. This is useful for multiple interface elements that may have similar names, but are not exactly the same. "Untitled-1", "Untitled-2", etc., can be represented by the regular expression "/Untitled-≈/".

When running against multiple target machines, agents can communicate with each other by sending and receiving messages. This allows synchronization of events for software that can be run on multiple systems at the same time. This is useful for network software products that must be installed on more than one machine to communicate with one another.

The best place to learn about the features of VU is to look in the VU Language Reference manual. It is well written, and thoroughly describes the capabilities of the VU language.

Summary

Automated testing tools will never be able to entirely replace human testers in QA, nor should they. Most of what QA does is about discerning how a user might interact with a product, and a computer cannot always predict these actions or define a good way to test them. However, used properly, automated tools can greatly increase the productivity of QA, saving labor costs, increasing consistency, and shortening time to market. In today's highly competitive environment, automated tools can provide you with the edge you need to succeed.

[*There is a great deal of VU-related material on the Tool Chest editions of Apple's Developer CD. For instance, in the May CD, which was the current Tool Chest CD as this issue went to press, the Testing & Debugging folder inside the Subject Index folder held aliases to various folders containing VU material, including a tutorial and a host of tools that extend VU's abilities in valuable ways (note, in particular, Ivy, which allows VU to "capture and compare screen images"). – man*]

WWDC

By Matt Neuburg and others

Looking for the Future

What did you learn, Dorothy, in the Land of Oz?

Special thanks to our on-the-scene experts – John Clements, Chris Magnuson, Jim George, and Jeremy Roschelle – who gave selflessly of their time and expertise to provide us with reports, only to have them hacked to bits and buried in this article. We owe them for much more than the small snippets explicitly quoted here.

Another WWDC has come and gone, bombarding us with four and a half days of talks by Apple employees (and others), accompanied by large projected images, consisting mostly of: (a) the magnified talking head of the speaker; or (b) cryptic text summaries (see Figure 1); or (c) software demonstrations ranging from the suspiciously glitzy to the refreshingly crashy.

Assuredly, a genuinely instructional component is not entirely absent from the proceedings; and for this, of course, we're always grateful. Still, the presentational mode does have a certain mind-numbing uniformity; and one does occasionally get the sneaking suspicion that the event is really an elaborate excuse for the carefully orchestrated peppering of press-release announcements that have been so obviously timed and reserved to be released at intervals throughout the proceedings (see http://product.info.apple.com/pr/library/1996/may.html for a sampling).

At such an affair, hard information is like gold – highly valuable, but deriving some of that value from its scarcity. I felt a great sympathy for developers trying to make a living around the ramifications of Apple's elephantine movements, seeking hints of what those movements might be likely to be. As Jim George says, "The 'talk in the hall' was centered on analysis of Apple's technologies, plans, and tool offerings." The sad thing is that such analysis had to be based so heavily on speculation. My own personal picture of where Apple might really be heading, what technologies it will really pursue, is not much less murky than if I had never attended the convention.

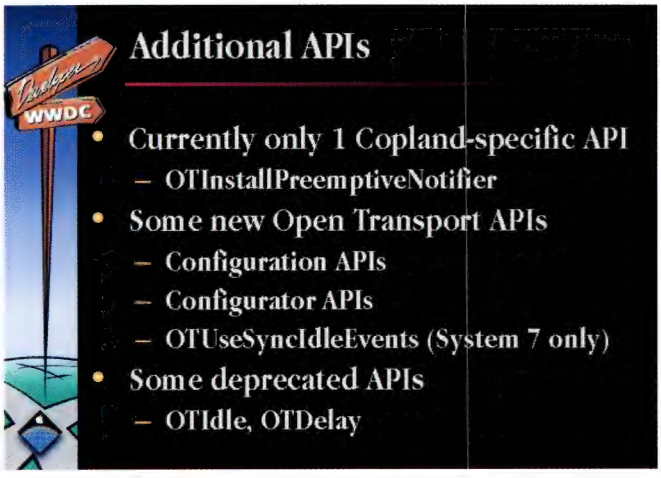


Figure 1. Get ready for forty-five hours of this (we particularly hope you like the logo at the left, because you're going to see it a lot)

Mac OS 8

The biggest draw (largest number of sessions held in the biggest rooms) was surely Mac OS 8. Clearly, in some ways, Mac OS 8 is appealing. I've been ranting for years about the stupidity of the event loop and the back-and-forth between system calls and program response required just to put up ordinary objects like windows and buttons, a legacy from the days when a Mac had small RAM and tiny ROM. In this regard, Mac OS 8 should be the answer to a prayer; the event loop is doomed, ordinary GUI objects are maintained by the system, and my code isn't alerted until there's an event to which I really have to respond, all of which should make the Mac a lot easier for "the rest of us" to program.

Amongst the critics, though, debates on architectural and market issues were rife. Do we really need a whole new system to do this? Or, just the other way: isn't Apple's drive for backwards compatibility holding Mac OS 8 back to a dangerous degree? Is pre-emptive multitasking really the best way to go? Isn't the much-touted memory protection model going to be a long time delivering benefits? Will the virtual memory model prove compatible with Apple's best existing technologies? Will end-users really see a speed gain, and when? When will the Mac OS 8 architecture and details start settling down into something stable and reliable, and stop mutating like a Transformer toy?

The best way to get a firm feel for Mac OS 8 as it exists now was to sign up for time in the hands-on lab. From here, there emerged some vivid reportage. John Clements writes:

> There was a hands-on demo of the current state of OS 8. There were tantalizing glimpses of the goodies to come, but the overall experience was awful. It does not yet support text editing, so you couldn't actually do anything except open and view documents (any dialog field that needed something typed into it was blank and dead). Also, it was incredibly fragile and crashed repeatedly, often corrupting system files on the disk in the process. The demo staff reformatted and rebuilt the hard disks at regular intervals. It was incredible that they even let us see the beast.

And from Chris Magnuson:

> One thing I discovered is that, with this build of the OS, development is difficult. The floppy drive wasn't working under Mac OS 8, so I had to build code on my Powerbook, then boot the test machine with System 7.5, copy the files from the floppy to the test machine, then reboot the test machine with Mac OS 8 and run the code. This long process was a limiter on how much I could get done in the time allotted.
>
> The application I was working with was a heavy Sound Manager 3.2 user. I had written this application over the last year and knew it inside out. The first thing I found was that the Gestalt call with the selector for a built-in sound input port wasn't working. In fact, the machine would hang. I commented this out and went from there. The next place it hanged was querying (using Gestalt again) to see if Sound Manager 3.1 or better was present. Obviously it was supposed to be, so I commented this out too in order to get on with the job.
>
> The next thing that I saw was visually shocking. One of my dialog boxes came up and the background was all grey (not white), with funny white regions around some of the dialog items (in particular, the sliders). I had been prepared for this mentally but nothing does it to you like seeing it. This dialog is going to take some work, because it uses custom defprocs (CDEFs) for the sliders. The assumption was made that the dialog background would be all white; this was now no longer valid. I will have to redo this code so that on Mac OS 8 a different slider will be used – one that is savvy about the Appearance Manager.

Internet and Java

Particularly noteworthy was the tendency to throw the incantations "Internet" and "Java" at everything like some sort of fairy dust. The public networked Macs were running Cyberdog, which seems to have been promoted from an OpenDoc proof-of-concept to some sort of killer-app wannabe (though the one I tried just crashed on me when I tried to send mail with it). Java applets were shown running inside OpenDoc and every other imaginable sort of container, and even poor old HyperCard seemed to be maintaining a lease on life only by promising that stacks would some day manifest themselves by way of a Web browser. Press announcements proclaimed distant sightings of Java on all horizons (Pippin, Newton, Mac OS), but I haven't developed any personal internal sense of what this might mean in practice.

Jim George puts an interesting spin on the Internet situation:

> Apple's future is not entirely in its own hands, but lies in "strategic" alliances and partnerships with many other hardware and software developers and suppliers. On the surface, Apple is more dependent on these alliances than the reverse; either the Mac versions already constitute only a small percentage of their market, or the companies are expanding into the "more lucrative" Wintel market. Yet, it is important that alternatives and competition continue in personal computing platforms, as a guarantee for continued innovation. The Internet phenomenon happened with little help or leadership from either the Wintel or the Apple market; in fact, both are being changed by the Internet!

OpenDoc

There were a great many OpenDoc sessions, and certainly these generated the most striking demos, because, by its very nature, OpenDoc consists of curious actions occurring in unlikely contexts. Most eye-catching of these was a spreadsheet (codenamed "Baywatch") by Adrénaline Software of Québec, which made a highly customizable animated three-dimensional graph out of its data – though this seemed to me an advertisement less for OpenDoc than for QuickDraw 3D, a technology of whose brilliance no one should need any convincing.

Once more, Java and the Internet were the props most heavily relied upon. The legerdemain included Netscape plug-ins or Java applets made to run inside OpenDoc, intimations of integration between Java and OpenDoc or Java and SOM, and, of course, poor old Cyberdog.

This analysis comes from Jeremy Roschelle, who, as long-time readers of this magazine will agree, ought to know:

> The announcement that Netscape would become OpenDoc-compatible, along with the adoption of OpenDoc by the Object Management Group, is important because it means that CORBA distributed-object computing is going to have a home in Netscape through OpenDoc.
>
> On the C++ framework front, Metrowerks showed their OpenDoc/PowerPlant bridge. "PowerPart" is now shipping on DR/9 (with a little help from yours truly).
>
> Digital Harbor was showing WAV, a very cool OpenDoc-based "work processor" with an innovative "task bar" (http://www.digitalharbor.com/docs/wav.html). WordWrite also announced their intention to be a container app. Apple had their first set of "QuickStart" components that cover each standard Mac OS media type. Expect to see a lot of components by Macworld Boston.
>
> On the cross-platform front, IBM said they have a Windows OpenDoc beta nearing completion, and the 1.0 releases of OpenDoc for Windows95 and WindowsNT should be completed this year (http://www.software.hosting.ibm.com/clubopendoc/tools.html). Apple is preparing ODF to compile to Windows "as fast as they get stable code from IBM".
>
> On a technical level, the most significant announcements were related to OpenDoc under Mac OS 8. As I've long said, OpenDoc requires a better memory manager. Under Mac OS 8, documents will have no `'SIZE'` resource, and in fact will not consume an application partition at all. There will be no fixed-size heap. Instead, the memory manager will allocate RAM and virtual memory on demand, until you fill your hard disk. This way, your doc will run no matter how many components are embedded. To be compatible, use only the OpenDoc Memory API calls.
>
> Mac OS 8 will allow parts to launch pre-emptive tasks (e.g., for computation and communication). Bento, the OpenDoc storage architecture, will be highly optimized. Apple event encoding and decoding will be optimized by what sounds suspiciously like Jen Alfke's AEGizmos. This will become part of the OS. SOM and Java will both be deeply integrated in Mac OS 8, too. The goal of the OpenDoc team is to be able to launch a document in two seconds or so. Let's hope they achieve this, if for no other reason then to make iterative debugging faster.
>
> Finally, at the human interface, Mac OS 8 will support OpenDoc seamlessly (Kurt Piersol is now assigned to that job). There will be OpenDoc viewers for every media type. Pop-up folders will support part stationery very gracefully. A new titlebar will make it easy to drag and drop an entire document. Perhaps most importantly, the new HIObjects (which replace the Window, Menu, Control, Dialog, and TextEdit Managers) will gracefully install into any OpenDoc facet.

Jim George adds a more sombre note:

> Claris's approach for ClarisWorks and OpenDoc is interesting and provoking. They analyzed ClarisWorks, their installed base, the OpenDoc architecture, and their business model, and concluded that implementing it as OpenDoc *parts* that the user could configure at will did not fit well with their business marketing/delivery model. So they decided to upgrade ClarisWorks to become a OpenDoc *container*. Further, they found that CALib was not the tool that was needed, and developed a tool, the Claris Container Library, which will be made available to developers; and Claris will upgrade it until the end of '96.
>
> OpenDoc component parts as a business model (i.e., how to make money with parts) is not understood: if Apple cannot even convince Claris to adopt the OpenDoc software model, how will they convince other developers?
>
> To date, OpenDoc is available for Mac OS, OS/2, Windows (alpha) and AIX (beta). But this is not the ordered list that developers needed – no final product for Windows or UNIX with an appreciable installed base until mid-1997. Keep in mind that AIX is not the UNIX employed by high-end educational and scientific research laboratories; for them, SunOS, Solaris, and SIG are UNIX!

And So, Until Next Time

Apple continues to reorganize itself and its plans, so WWDC can be only a glimpse of where Apple might be at a particular moment in time. The glimpse we've presented here is limited also by our own resources; a full report would require the whole magazine, and besides, you just had to be there to enjoy the full cafeteria of pies in which Apple has a finger. Of course we'll do our best to be informative about a wide range of Apple technologies, in our regular articles. If you want more information about what WWDC was like, you can see a sketch of the timetable for each day of the convention at http://www.info.wwdc.carlson.com/cmg/day1.html (and day2.html, and so on); and http://wwdc.carlson.com/ tells you how to purchase a CD of what you missed.

FROM THE FACTORY FLOOR

By Dave Mark

Andreas Hommel, Compiler Architect

This month's interview is with Andreas Hommel, one of the original minds behind Metrowerks' compiler architecture. (See "A Little CodeWarrior History", *MacTech Magazine* 12.7 [July 1996] 61–64, where John McEnerney recalls being shown Metrowerks' newly acquired C compiler which "a guy named Andreas Hommel in Hamburg had been writing ... as a hobby".) You'll meet a pretty interesting person and, at the same time, learn a thing or two about the compilation process.

Dave: How did you hook up with Metrowerks?

Andreas: I got interested in compiler construction while I was still in University. I was writing computer games, and most C compilers didn't really produce very good code. Also, I liked ANSI C a lot, but back then, Mac compilers didn't really conform to this standard. So at one point, I decided that it would be fun to write my own compiler in my spare time. Two or three years later, I had my own little IDE and an ANSI C compiler with some C++ extensions.

I was about to finish my CS degree, and I had a good desktop publishing job offer in Hamburg, but I really liked working on my compiler project, so I decided to give it a try. I sent out a bunch of demo disks to some Macintosh compiler-related companies. A few months later, Greg Galanos called me and we started talking about technical details of the compiler and how we could come together. After a couple of Transatlantic phone calls, Greg invited me to come to Montreal to meet with him and Jean Belanger. They spoke about the incredible opportunities for a compiler company, given Apple's pending transition to the PowerPC chip. We also talked a lot about all the technical aspects of the compiler and how it could be changed to support another code generator and a Pascal front-end. A week later, we had signed a contract.

The next 6 months were pretty busy. I still had some work to do for my old job, I had to finish and defend my thesis, and I had to start moving the compiler towards C++ for Metrowerks.

Dave: For folks who've never written a compiler, can you describe the compilation/link process?

Andreas: The compiler transforms each individual source file (or "translation unit", to be technically correct) into an object file. The functions, procedures, and variables in a source file are transformed into code and data in the object file. The code in an object file is not executable because it usually contains references to code or data in other object files; these references have yet to be resolved by the linker. The unresolved references are also stored in the object file. The CodeWarrior IDE actually hides much of this process, because it stores all the object files in the project file, so you don't see them on your hard drive (if you use the CW MPW tools, you will actually generate individual object files). The object file also stores symbolic information that is used by

Andreas Hommel is currently the C/C++ front-end and 68K back-end/linker architect at Metrowerks. After finishing his Master's degree in Computer Science, Andreas did some contract programming in the desktop publishing area and also published several games on the Macintosh and Amiga. He has been with Metrowerks for three and a half years.

Andreas lives in a small country village about 20 kilometers north of Hamburg, Germany, with his wife, two Australian Shepherd dogs, and two Arabian horses. When he is not coding, riding horses or walking his dogs, Andreas likes traveling, playing a good video game, and driving really fast on the Autobahn. He also likes cooking and fine red wines (California Cabernets in particular).

Bigger, Better, Faster!

Introducing the new MacTech CD-ROM™ Vol. 1-11. Get all 127 issues of *MacTech Magazine*™ plus a new THINK Reference™ for faster access, all the source code, working applications, demos, frameworks, and more – all in one CD!

- 1420+ articles, from all 127 issues of *MacTech Magazine* (1984 – 1995).
- Improved hypertext, and a new THINK Reference Viewer – for lightning-quick access.
- New super-fast word search.
- 100+ MB of source code – use them in your own applications, with no royalties.
- Full version of THINK Reference 2.0. The original online guide to *Inside Macintosh*, Vols. I-VI.
- 80MB of FrameWorks/SFA archives. The most complete set of FrameWorks archives known.
- Sprocket™! MacTech's Tiny Framework that compiles quickly and supports System 7.5 features.
- The best threads from the Macintosh programmer newsgroups, plus thousands of notes, tips, snippets, and gotchas.
- Popular tools that Macintosh programmers use to increase their productivity.

Available Through Developer DEPOT™

Now Only $89!

$39 for upgrades from previous versions.

(prices do not include shipping and handling)

http://www.devdepot.com • Phone: 800-MACDEV-1 • 805-494-9797 (outside the U.S. and Canada)
Fax: 805-494-9798 • E-mail: orders@devdepot.com

the Source Debugger to map source code to executable code and to find variables and their types. If you are interested in this, and you know a little bit of Assembly Language, you can use the **Disassemble** command (in the **Project** menu), which will generate a pretty complete object file dump for a particular source file.

The linker then takes all the object files (a library file is basically just an object file), resolves the external code and data references, and generates an executable file from that. On the Macintosh, the linker also merges your application with the resource data and generates a SYM file that is used by the debugger.

Dave: What about the actual compilation process?

Andreas: The compiler itself can be grouped into several phases. A user typically doesn't notice these individual phases, and in fact most compilers do not strictly execute one phase after the other, but this logical grouping really makes it a lot easier to implement a compiler.

The first phase is the Lexical Analyzer or Scanner. This part of the compiler "looks" at your source code and splits it into individual tokens. A token is a small lexical element. For example, in C, operators such as `'+'`, `'--'`, `'->'`, and keywords such as `for` and `while` are individual tokens. Identifiers, numbers and strings are also considered as tokens with attributes. For example, `"123"` is a numerical token with the attribute/value 123. A lexical analyzer for C and C++ is quite complicated, because it usually also implements the preprocessor, so it has to take care of source file inclusion (`#include`), macro expansions (`#define`) and conditional compilation (`#if`). All this is hidden from the remaining parts of the compiler, and the CodeWarrior lexical analyzer transforms a source file into a uniform stream of tokens.

The next phase is the Syntax Analyzer or Parser. Most computer languages (including C, C++, Pascal and Java) have a grammar which is a set of rules that describes which token sequences form a legal program. The parser makes sure that the stream of tokens conforms to these rules. For example, the rule for a `while` statement is:

```
<iteration-statement>: while ( <expression> ) <statement>
```

`<iteration-statement>` is the name of this grammar rule. `while`, `'('` and `')'` are tokens; `<expression>` and `<statement>` are called non-terminal tokens, which means they have to be replaced by other tokens or rules. The parser transforms the stream of individual tokens into another data structure (usually a syntax tree) that is used by the remaining phases.

The next phase is the Semantic Analyzer. The parser makes sure that a program conforms to the grammar, but it doesn't catch any other types of error. For example, "`1=2;`" is a *syntactically* correct C assignment expression statement that is *semantically* incorrect, because you cannot assign 2 to 1. So this phase makes sure that types in a program match, that all identifiers (or variables) have been defined, and that operand types in expressions match.

Now we have a legal program and all we have to do is generate code from it. One could generate code directly from a syntax tree, but usually a compiler generates an intermediate code representation. For example, CodeWarrior uses a tree-based intermediate code (IR tree) that is very close to a syntax tree but has a lot of additional information about types. In fact, the CodeWarrior compiler folds the syntactic, semantic and intermediate code generation together, so basically the parser also checks the semantics and it generates an IR tree. This really speeds up the whole process. This IR tree is really the key to our compiler technology. All code generation is based on this tree. So this makes our front-ends (C/C++/Pascal) and back-ends (68K, MIPS, PPC, x86) interchangeable.

This IR tree is then passed to the individual back-ends where it is used to generate the actual 68K, PPC, MIPS, or x86 code. The back-ends are all pretty different, but they all transform the IR tree into machine instruction sequences, allocate memory and registers for variables, and generate an object file. All back-ends also do some machine-level optimizations (peephole optimizations) that replace instructions with better ones or remove redundant instructions.

We have an IR-level optimizer that removes redundant parts from this tree and does some other basic things to optimize branches. We also have a high-level IR optimizer that is currently used in the x86 compiler, but this optimizer will eventually also be used in the MIPS or 68K compilers. The PPC back-end is a little special because most of its optimization is done in the back-end. This makes sense, because the PPC's RISC instructions are very simple, so you can do a lot of high-level optimizations (such as loop optimizations and common subexpressions) with more fine control on the actual machine level and get better results. This would be very hard to do for a CISC processor like the 68K with all its complex instructions and addressing modes.

Dave: With that in mind, what did your original development environment look like, from a technical perspective?

Andreas: It had pretty much all the basic functionality you need to write programs in C. It had a project window, a multi-window text editor, find and replace, and some simple Preference dialogs. It even had some nice little features such

as function popups and multiple access paths, but a lot of major features such as multi-language support, plug-in compiler support, collapsible project views, syntax coloring, split-pane editing, multiple-pane Preference dialogs, a tool bar, and tool-server support, have been added to it since then.

Dave: How would you compare your original compiler architecture with the current CodeWarrior architecture?

Andreas: The original compiler and linker didn't support a plug-in interface, and everything had to be linked into the IDE. I always had multiple back-end support in mind, and I was always using the IR tree. However, when John McEnerney started writing the PPC back-end, I had to clean up the front-end/back-end interface, and we had to change a few other minor things. There were also some changes in the 68K back-end, to support some Pascal-specific features such as sets and nested local variables.

There have been a lot of changes in the front-end. The original compiler had some basic support for C++ classes and function overloading, so a lot of stuff had to be added since then. I also had to change quite a few things to support multiple platforms (Mac OS, x86, MIPS). Recently, I had to make some additions to the IR to support zero runtime overhead exception handling. This also required changes in the back-ends.

Dave: What special work do you have to do in the front-end to enable multiple platform support? For example, what did you have to do to CodeWarrior to make sure it would support x86, MIPS, and 680x0 code generation?

Andreas: There are a number of areas where a front-end needs to be aware of back-end requirements. For example a C `struct`'s member layout is done in the front-end, so you have to be aware of the data-member alignment of the target architecture. Another problem has to do with integral and floating-point type differences. For example, a `long double` is an 80-bit type on 68K but a 64-bit type on the PPC. In the same vein, the x86 uses a different byte ordering (little-endian) than the 68K (big-endian). Most of the functions that deal with these issues have been isolated into target-specific front-end files.

There are also some language-specific issues. For example, both Apple and Microsoft have their own C and C++ language extensions, and the front-end needs to support all of them. One of the biggest problems is C++. There are many very powerful features in C++ like multiple inheritance, polymorphism, exception handling and runtime type identification. The ANSI C++ Standard defines how all these features work, but it does not define how they should be implemented, so every compiler vendor has their own implementation. For example, there are many ways to implement virtual function calls or to allocate base classes in a derived class hierarchy. We always had our own implementation for this, but now, with the x86 compiler, we also have to conform to Microsoft's standard class layout. We're still targeting full compatibility, and this requires more work in the front-end as we go forward.

Dave: Beginning with CW8, CodeWarrior offered support for zero runtime overhead. First, what is zero runtime overhead? Second, what advantages does it offer?

Andreas: C++ exception handling requires that all local class variables that have been constructed between a `try` and a `throw` be destroyed before an exception is handled in a `catch` block. This process is called stack unwinding. Our first exception implementation was based on a pseudo-`setjmp/longjmp`, and a linked list of all active local stack objects needing destruction. This was relatively easy to implement in the front-end, and it required no changes in any of the back-ends. So we were able to support exceptions in all our back-ends at the same time.

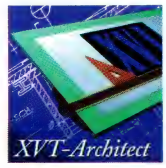
However, this implementation requires some runtime overhead. For example, each local class object that needs destruction has to be registered when it is constructed and unregistered when it is destroyed. Also, the `setjmp` call that was required for every `try` block was very expensive on the PPC, because it had to save all processor registers in a local buffer, and this implementation had to modify a global variable, so it did not work very well with threads.

Our zero runtime overhead implementation is no longer using `setjmp` or a linked list. Instead, the stack unwinding process is done by examining the processor's stack and using a pretty complex exception table that tells the exception handler how to locate local variables and how to destroy them. So there is no runtime overhead required for saving registers or registering local objects. Throwing an exception is actually a little bit slower because the stack unwinder is a lot more complicated, but usually exceptions are really exceptional so your application runs much faster. Another advantage is that this exception model doesn't have to modify any global variables, so it is really thread-safe.

The only disadvantage is that the stack unwinder can only unwind functions that do have an entry in the exception table. The current Mac OS doesn't have any exception tables, so you can no longer throw an exception in an OS callback function and catch it in your main program, which was possible (but not recommended) in the previous implementation.

Continued on page 62

TOOLS OF THE TRADE



By Edward Ringel

XVT DSC++

Cross-platform compatibility, but at a cost

Development, Money, and Reality

As most of us are painfully aware, the Mac OS does not exactly hold a dominant share of the market in personal computers. Therefore, just as a matter of simple economics, anyone who wants to make high-end profits selling software needs to support more than just the Macintosh platform. Furthermore, corporate MIS gurus prefer that users, if they must employ various platforms, should still not only use the same apps but also see similar interfaces – rightly, since this reduces training and support costs.

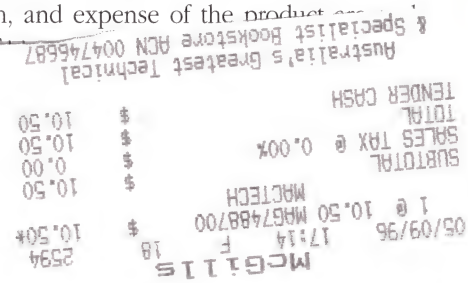
It's hard, though, for one person to write commercial-quality native code for multiple platforms; and it's expensive to have multiple talented programmers for multiple platforms. Consequently, the freelancer with the killer app and the in-house or contract programmer facing a deadline seek the same Holy Grail – write the code *once*, put it through a bunch of different compilers, come out with applications that work the same way on a variety of platforms, yet preserve the native look and feel of each host platform.

The challenge of making this possible is taken up by XVT Software, Inc., of Boulder, Colorado (www.xvt.com). The resulting product, DSC++ (Development Solution for C++) is a rich environment, but one that may not prove to be the right answer for every developer.

Some Preliminaries

This is an expensive and complex product, and I wanted to evaluate it from within a clearly defined mindset, partly to introduce order, and partly because my comments might influence others' purchase or evaluation decisions. Accordingly, I played the role of an experienced Macintosh procedural programmer, ready to migrate to C++ and a framework. I imagined that I had a contract that would ultimately need porting to a second or even third platform, and that therefore I would need a cross-platform tool; I was evaluating DSC++ with intent to purchase.

The depth, breadth, and expense of the product are [illegible] that, if purchased, DSC++ [illegible] the expertise, [illegible] would have a [illegible] Accordingly, thi [illegible] framework, but [illegible] library that impl [illegible] compatibility, [illegible] application cons [illegible] with technical a [illegible] information is cri [illegible] in this rapidly cha [illegible]



I reviewed DSC++ version 3.22.01 for a 68K Macintosh. The product was installed on a Performa 6200CD, and CodeWarrior 8 was used for builds and perusal of code. The carton weighed fourteen pounds, and contained several books, a few papers, and 7 HD disks. Review of the books is a good place to start, as the printed materials reflect the components of the system:

Ed Ringel is *MacTech Magazine*'s new Contributing Editor for Product Reviews. His interest in things Macintosh reaches into the dim past of Fat Macs and 400K disk drives. When he's not programming or enjoying the Maine lakes and woods, he practices respiratory and critical care medicine in Waterville, Maine. He can be reached at eringel@mint.net. Contact him if you're interested in joining *MacTech*'s product review team.

Tools.h++ Introduction and Reference Manual. Instructions for the use of Tools.h++, a collection of useful foundation classes for data management and manipulation. Used throughout the C++ product.

Guide to XVT Development Solution for C. Guide to the C function underpinnings of the C++ framework and library and the resource language.

Guide to XVT Development Solution for C++. This is the guide to the C++ application framework and the class library.

XVT Portability Toolkit Reference. Comprehensive reference to the function library common to all XVT products.

XVT Power++ Reference. In-depth reference to the class library for DSC++.

XVT Architect Manual. Manual about and tutorial for XVT Architect, the application construction tool for XVT Power++.

XVT Platform-Specific Book for Macintosh and Power Macintosh. Some specifics regarding implementation of XVT products on a Mac OS platform.

XVT Development Solution for C++ Quick Reference; XVT Development Solution for C Quick Reference. Two small books that prototype classes, structs, enums and functions.

The published materials are of the highest quality. They are visually pleasing, physically well manufactured, and very well written. The writers use a consistent, clear, systematic approach to describing functions, classes and functionality. It was very easy to find and learn about material that I wanted to study. In particular, Rogue Wave's Tools.h++ manual is among the best pieces of technical writing that I have encountered. The sole exception is the *XVT Architect Manual*, which was not quite as clear and not at the same depth. Additionally, some of the tutorial code in the *XVT Architect Manual* did not match exactly that generated by the XVT Architect. However, I could still follow along pretty easily.

Installation of DSC++ was a disappointment, and the first indication that I would be dealing with a product that did not follow the Macintosh Way. The environment comes as a seven-disk Compact Pro self-extracting archive. After performing the extraction, I looked somewhat unhappily through the various folders with their cryptic, UNIX-type designations (pwr, ptk, shell, bin, for instance.) One could argue that retaining this naming convention is a constraint imposed by the cross-platform compatibility requirements, but obviously it defeats one of the nicest features of the Mac OS. I then spent ten minutes sifting through these folders, pulling out several megabytes-worth of Symantec and MPW files unrelated to my CodeWarrior environment. Compact Pro is not an appropriate installer for a complex multi-option installation. (As implied, XVT DSC++ will support CodeWarrior, Symantec, and MPW IDEs. Support for Symantec THINK products was dropped as of March, 1996.)

Content and Structure

My next task was to correlate documentation with files, and to try to develop some mental apprehension of how the pieces fit together (I had to "get it"). The excellent documentation made this fairly easy.

XVT Portability Toolkit

XVT makes two closely interrelated products, a cross-platform development environment for C, and another for C++. Mac OS, Windows, Windows NT, Sun, and other UNIX platforms (essentially any OS that has a GUI) are supported. XVT has a collection of C functions, the XVT Portability Toolkit, that acts as a uniform interface to the various native GUIs. Thus, to draw a rectangle in the current GrafPort with a 1-by-1 pixel pen, as we might on the Mac with this code:

```
Rect myRect;
SetRect(&myRect, 10,10,100,100);
PenNormal();
FrameRect(&myRect);
```

we write instead:

```
RCT myRCT;

/* RCT is a rectangle structure that is identical to a Rect */

xvt_dwin_set_std_cbrush(theWindow,TL_BRUSH_WHITE);

/* need to set the "brush" to paint the inside of the rectangle. theWindow is assumed
to be previously defined, and is a long. TL_BRUSH_WHITE is a macro that fills in the
fields of a CBRUSH struct, which is a descriptor of the painting environment. */

xvt_dwin_set_std_cpen(theWindow,TL_PEN_BLACK);

/* need to set the pen; TL_PEN_BLACK is a macro that fills in the fields of a CPEN
struct */

xvt_rect_set(&myRCT,10,10,100,100);
/* same as the real thing */
xvt_dwin_draw_rect(theWindow,&myRCT);
/* ditto */
```

Functions use a uniform naming convention. An XVT call always starts with `xvt_`. The next "word" specifies the general category; in this case it is drawing in a window, `dwin`. The next word specifies the action, such as `set` or `get` or `draw`. Finally, if appropriate, the last word or two indicates the actual object in question, such `rect` or `cpen`. The function `xvt_dwin_draw_rect`, then, is virtually self-explanatory. Of course, before I could even begin to use the product, there would be a whole series of new macros and typedefs to learn. However, this is part of the price of admission to any comprehensive class library and/or framework system. The ability to use the same call with the appropriate result on a Mac, PC, or Sun workstation is obviously the core of this product.

It is important to emphasize the word *appropriate:* the XVT Toolkit is a layer which interacts with the native toolbox, so that a window on a Mac looks like a Mac window, a scrollbar in Windows looks like a Windows scrollbar, and so on. This permits the cross-platform application to meet the interface expectations of each platform's users. The Portability Toolkit covers a wide variety of operations, though still just a subset of the Macintosh feature set. Features pertaining to the application, clipboard, controls, drawing, windows, dialogs, fonts, texts, images, errors, memory, iostreams, lists, printing, rectangle utilities, and cursors are all supported.

Of course, there may be an efficiency hit. Since all I had are binary libraries I don't know for certain, but I suspect (given

the `theWindow` parameter in `xvt_dwin_draw_rect`) that the implementation contains code that on a Mac either tests for current `GrafPort` or does a `SetPort()` before drawing, adding overhead to every drawing call. Normally, I can `SetPort()` when I want to, and just do high-speed drawing after my `GrafPort` is designated. There's no XVT equivalent.

The dynamic throughout is thus one of well documented, well thought out functions that might well be less efficient than native Toolbox calls. (The XVT Portability Toolkit does not exclude the use of native calls. It is possible to extract `WindowPtrs`, `ControlHandles`, and the like, and use them freely. It is possible also to intercept the event loop and process a raw event. However, once you make the code platform-dependent, it is obviously no longer 100% portable.) And, of course, these functions weren't what I was used to. Facile use will require an extensive re-education of the programmer. The prospective purchaser must balance the drawbacks of all this against the problems of supporting multiple platforms using multiple native development environments: this is the crux of the purchase decision.

URL and Curl

GUI calls in the absence of some description of the interface make for a cumbersome programming task. The Macintosh concept of resources is appealing, and XVT uses a Universal Resource Language (URL) to describe the GUI components. This resource language is cross-platform portable. In each development scenario, the interface description is translated into the appropriate platform-specific resource descriptor. On the Mac, a URL file is turned into an .r file by **curl**, an MPW tool which I ran under ToolServer. I then used MPW Rez, still under ToolServer, to create my resource file. There is also a freestanding app, called curl.app, but scripting and the CW environment made curl much easier to use. curl.app was very slow, had a poor interface, and was not user-friendly. (As an example, in order to set access paths for `#include` files for this program, I needed to type full path names into a specially-named text file.)

The resources described are, again, a subset of the Mac feature set. Window, control, dialog, string, picture, menu, and icon formats form the basic kernel of interface development. URL resources can be created by text coding as with Rez, or by drawing with the XVT Architect application, discussed below. (The companion C product has an application for drawing the interface as well.)

XVT Power++

XVT Power++ is the class library and application framework produced by XVT. The class library and framework have similarities to and differences from the "big three" on the Mac scene (MacApp, TCL, and PowerPlant). The biggest difference, of course, is the absence of reliance on native Mac calls for execution of each method. Instead, a combination of native Macintosh calls (in the binary libraries), C++ methods designed specifically for DSC++, and XVT Portability Toolkit C calls are used for implementation of the functions. Because of this structure, there is often an extra calling layer between C++ and the Macintosh Toolbox.

The general organization of the framework is that of the model-view-controller popularized by Smalltalk. Each application has a single application object which is at the apex of the hierarchy. This object is responsible for opening and closing documents, initializing the application, and the like. The application object has methods for doing saves, changing fonts, etc., which to my recollection is somewhat different from other frameworks on the Mac. Inheritance is generally single (like MacApp), rather than multiple (like PowerPlant). There are a lot of pros and cons to this approach, and while multiple inheritance is elegant and parsimonious, single inheritance is clearer and easier to learn. I think particularly for a cross-platform product, single inheritance is a safer, more appropriate approach.

The application creates a document, which controls the data seen both within a view and in a file. XVT's document diverges from TCL and PowerPlant documents in the ability to incorporate Automatic Data Propagation (ADP) into applications. This feature allows changes in data shared among views to be automatically propagated; when operations in one view change data, the change is reflected in the linked views automatically. It's a bit like having intra-application publish-and-subscribe, which is certainly of considerable value. It is possible to create document-view relationships of significant complexity using ADP. The documentation and explanation of this aspect of XVT Power++ is very well done.

The GUI of the application is handled by view classes and subclasses. Although the nomenclature is different, window, view, pane, scrolling view, and control classes are all supplied. There is a fairly wide variety of graphical elements from which to set up your application. Often the view at the bottom of the hierarchy is a "native" view, which actually implements the Mac window, scrolling list, text edit field and the like. One very nice feature of the framework is the capability to attach an environment class to a view, window, and/or the application. The environment class automatically defines drawing characteristics for that object, setting the pen, brush, background, font, etc., when that object is the focus of drawing calls. The elements provided permit the production of a typical Macintosh look-and-feel program of early 1990's vintage. There is no support of QuickTime, sound, etc.

XVT's commitment to run on multiple platforms adds the need for a Task Window (the window in which the application runs), which is not necessary on a Mac. This is pretty much transparently handled for you when you use the DSC++ product. Multiple platforms also preclude the tight integration of Apple event handling at the class level. All in all, the framework and class library are very creditable from a structural standpoint, and will permit implementation of a broad range of functionality, but without any of the niceties unique to the Macintosh. As with the Portability Toolkit, the documentation

is excellent. This is a good class library comparing very favorably with the "big three".

Hooray for Rogue Wave!

In my opinion, the best part of the product is the Tools.h++ Class Library, a set of utility, collection, and storage classes, supplied as headers and object files, from Rogue Wave. It is possible to implement a wide variety of memory and file structures very simply just by deriving subclasses from those supplied with the library. Lists, unordered collections, hash tables, b-trees, strings, and many, many other very useful objects, coupled with the framework, make it possible to create and maintain complex data structures with DSC++. Since the data structures of Tools.h++ are the underpinning of the XVT Power++ data structures themselves, integration is easy and reasonably intuitive. There exists in the Rogue Wave library the ability to create data structures which are platform-independent as well, implying cross-platform file compatibility. The Tools.h++ manual is wonderful and an education in intelligent C++ programming. These high-quality utility and storage classes really make this product stand apart from the "big three", and are among the most compelling arguments for purchasing this product, cross-platform issues notwithstanding.

The strength of the data management capability of the framework reflects the workhorse nature of this product. It is very much a system for facilitating access to data and for managing the GUI to that data. This is almost certainly due to the fundamental data orientation of many of the other platforms supported by XVT, for which the GUI is ultimately secondary. I suspect that many of the corporate clients demanding cross-platform applications care far more about efficient data manipulation than the latest GUI features, Apple events, or QuickTime, and that the GUI offered by XVT is more than adequate for very good corporate in-house applications.

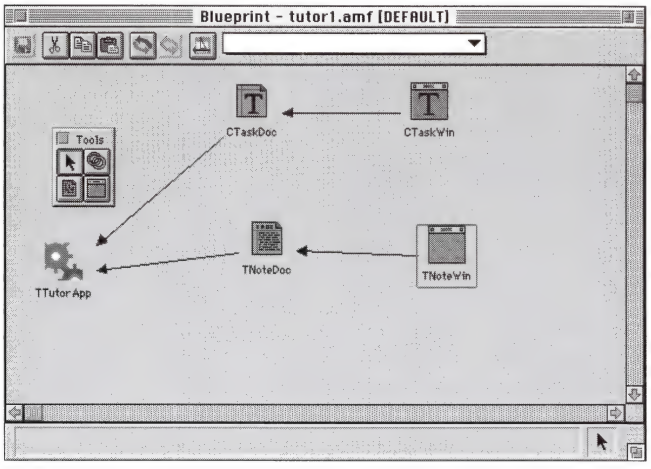


Figure 1. Blueprint window for XVT Architect. The application, documents, and windows are represented as icons, which are instantiated by choosing the document or windows tools. These objects are linked using the linking tool.

XVT Architect

XVT Architect is the application construction program supplied by XVT, permitting the programmer to draw the application. XVT Architect has three different modules, referred to by XVT as "interfaces". Initially, in Blueprint interface, the programmer defines the relationships among the different documents and windows, and thus defines the application's runtime object hierarchy. This is done by instantiating an icon of either a document or a window, and then connecting the objects with a linking tool.

Drafting interface permits the programmer to draw views, windows, and menus and the various control and text entities. The various classes available in XVT Power++ are available as drawing elements in XVT Architect.

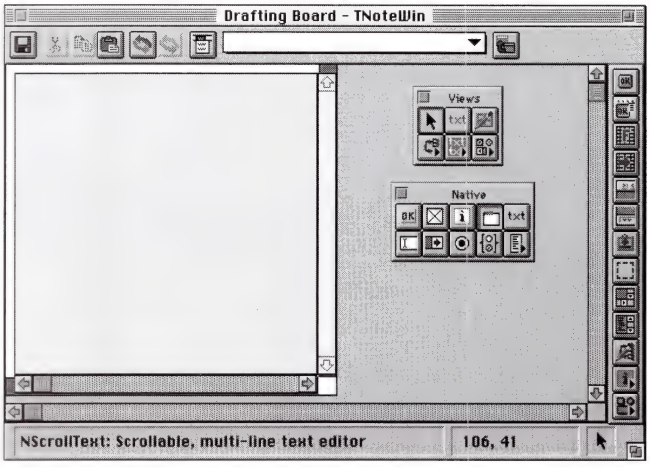


Figure 2. Drafting window for XVT Architect. The window under construction is to the left. Palettes of views, text types, and controls have been "torn off" so that they may be easily used.

Strata interface permits the programmer to set the characteristics of the various objects and classes. For example, if you wish to set the editable text view you've just created to use the parent window's scroll bars, this would be done in the Strata interface. It is possible to set all of the characteristics of all of the ancestor classes of the object in development. The characteristics are grouped by class derivation. In this respect, the Strata paradigm is particularly easy to use, as you can mentally break down the task into its component parts much more easily.

XVT Architect is slow and requires a 6MB partition to run reasonably. The factory-set partition on XVT Architect was not set correctly. Many of the design elements (such as the tear-off menus) are foreign-appearing. There was less than adequate attention to design: captions intruded into editable areas, and so on. I unintentionally created a low memory condition and XVT Architect handled this poorly; it did put up an alert that I had a memory problem, but then it corrupted its heap and ultimately crashed my computer before I could do anything to recover from my error. In one operation, clicking a button

http://www.mactech.com

(Need we say more?)

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

brought up a modal dialog that was positioned so that it was partially off the screen; given that I have a 15-inch screen, this is inappropriate. As a member of the over-forty crowd, and getting closer to bifocals than I want to admit, I was grateful for 12-point Chicago as the font of choice; however, by current standards of Mac design and aesthetics, it was unappealing. There were a number of areas where the ability to resize windows was ill-considered, and some of the visual clues that Mac users are used to were missing.

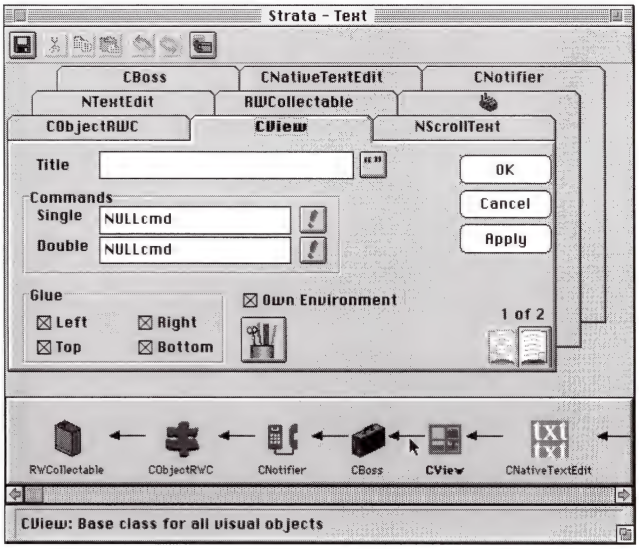


Figure 3. Strata window for XVT Architect. The class hierarchy for a scrollable text view is shown. The behaviors of the object can be set with this tool. All parent class characteristics can be set in each class's window. Along the bottom of the window, the hierarchy is represented iconically. Clicking on the icon brings up the relevant view.

When design is complete, XVT Architect will generate two sets of source code files. Much like Symantec's Visual Architect, XVT Architect creates "Factory" files, which create the objects and interface and are not meant to be modified by the programmer. These files are regenerated as the interface is changed. The second set of files is used by the programmer to implement the functionality of the application. These files are commented to show what code should be added or changed, and where. XVT Architect also creates a URL file and a script to automate the build of the project. To use this script under CodeWarrior, I had to add ToolServer to my environment (which I normally do not use), but once I got the configuration right, the build went forward flawlessly.

I have always had a love/hate relationship with interface builders. Whenever I see the line `//your code here` I cringe, because it requires that I enter another programmer's head and follow his or her way of doing things. Putting that bias aside, though, the concept behind XVT Architect is not bad at all, and XVT Architect uses the successful strategy of double file-generation to minimize enforced coding style. Certain aspects, such as the Strata interface, are particularly well thought out. Ultimately, Architect needs a lot more attention to detail and a big speed boost before I would be happy using it as a core development tool.

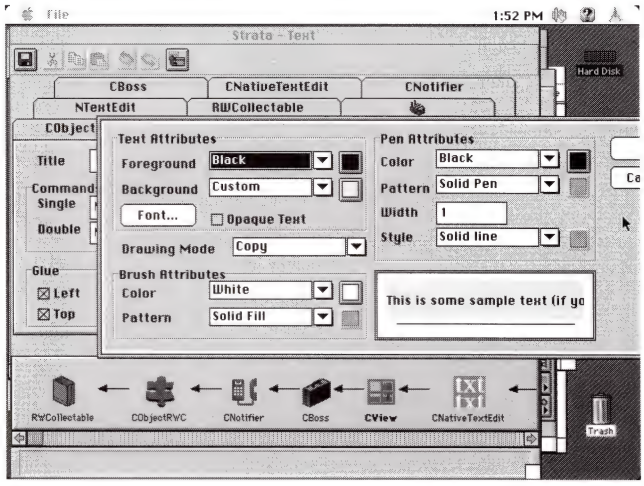


Figure 4. Oops! This XVT Architect modal dialog was placed partially off the 15-inch screen.

Code

I used example/tutorial code supplied by XVT as my test of compilation. Some operating systems do not permit file names longer than 8 characters. To achieve cross-platform compatibility, XVT `#defines` a descriptive macro for the name of each .h file, and the programmer `#includes` the macro rather than the true name of the file. This is great for Windows, but obviously unnecessary on the Mac. It also defeated CodeWarrior's mechanism for creating a pop-up menu of .h files, and I could not use the normal navigation techniques. I also could not get CW8's browser to parse projects. It was quite difficult to create a precompiled header for the projects, and I eventually needed to call tech support for help; even then I needed to modify factory-supplied .h files to achieve compilation of the header. The large number of compiler switches necessary for the various environments ultimately left some undefined compiler variables in the .h files which needed to be manually set. As pointed out by XVT tech support, there is apparently a bug in CodeWarrior 8 which does not permit proper handling of `static` variables in precompiled headers; XVT uses `statics` in its headers. I did not try to resolve this problem.

I worked through a couple of tutorial sessions using XVT Architect. The AppleScript script-directed build had set up my access paths correctly, and all files were available; I did not need to intervene manually to initiate compilation. The XVT Architect–generated projects compiled and ran flawlessly. In the absence of header precompilation, a simple scrollable, editable text window application required compilation of over

a quarter of a million lines of code. This same 68K application was over 600K in size and required a 2MB partition to launch without crashing.

I also compiled several of the example projects. These had been created without XVT Architect. I built these by hand, using curl.app to translate the URL file into an .r file, then using CW's Rez plug-in compiler to add the resources to the project, and finally compiling the C++ code as well. These projects all compiled without error, but at execution all failed initialization and aborted. They did not crash the computer. At this point, I was sufficiently frustrated that I did not try to resolve this problem either.

A number of the default behaviors of the objects are not appropriate for a Mac. For example, the demo applications appeared to have difficulty with mapping keystrokes properly, and again there were problems with placement of the windows on the screen.

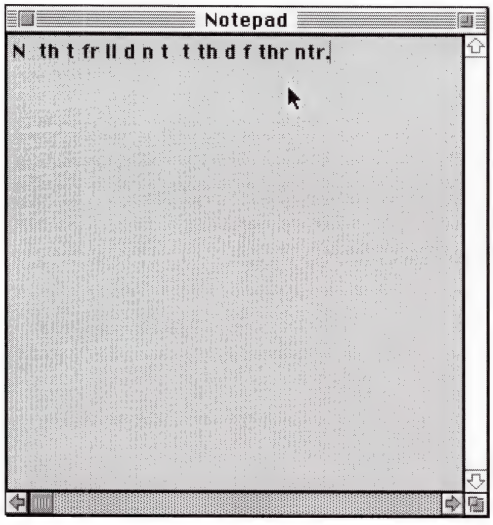


Figure 5. DSC++ handles keystrokes improperly. The compiled, fully functional tutorial contains a notepad window. The figure shows the results of typing "Now is the time for all good men to come to the aid of their country" into that notepad window.

The product comes with a number of Mac-specific extensions. As I stated previously, it is possible to implement Apple events, but this is not an integral part of the product, and has nothing approaching the support of PowerPlant, much less TCL. In brief, the XVT main event loop calls `AEProcessAppleEvent()`; all that needs to be done is to install the handlers. I am not an expert Apple event programmer, but it would seem that the structure needed to implement the events makes it hard to use the object model to create elaborate scripting and interapplication communication, but fairly straightforward to do some basic AE work. It is possible to retrieve handles to Mac data structures such as lists, editable text, and the like. It is possible to work with Mac resources directly, and, for example, directions are given on how to use Finder icons. There is a short manual devoted exclusively to Mac issues which is as well written and descriptive as the other written materials. The Mac-specific extensions are not all-inclusive, and the majority of the manual deals with resource issues.

Support

When I first received this product from *MacTech Magazine*'s offices, the code would not work out of the box: the CW7 libraries would not work under CW8 because, as many of you recall, this was a "recompilation required" update in the Metrowerks environment. I attempted to get customer support via email, giving the serial number on the master disk. I was courteously (but firmly) intercepted by a sales manager who had no record of a sale to me, and basically wanted to know what I was doing with this serial number. I explained who I was and how I acquired this copy, and so all subsequent contacts with the company occurred with their full knowledge of my status as a reviewer. Nevertheless, despite that knowledge and despite intervention from both the sales manager and XVT's public relations firm [*not to mention some well-chosen words from a certain magazine editor – man*], it took nearly two weeks before my status was resolved and I was given a password to their secure FTP site so I could download over 12MB of "patch". Obviously, as someone active in the software industry, I am very respectful of licensing issues; however, I would have hoped that, once I was identified as a legitimate user with a special problem, access issues could have been resolved more speedily.

There was, it turned out, no update or patch; the download was simply the same, complete product as on the distribution disk, updated for use with CW8 and with a few bugs taken out. It used the same Compact Pro archive. This archive was dated March 21, 1996, which was quite disturbing, as it implied (given Metrowerks' release cycle) that it had taken XVT in excess of two months to recompile their CodeWarrior libraries and to make them available for download (I had been told by the sales manager that the company was just completing this update when I first got in touch with them about this issue). While two months might not seem a long time, the Metrowerks (and Symantec) release cycle is four months, so companies supporting those development environments must occasionally expect to have to do some quick work to keep their customers happy; this, for good or ill, is part of doing business on the Mac platform.

Contact with tech support itself was mostly by email, and they were interested, helpful, and prompt. I dealt repeatedly with a single member of the tech support staff, and he (Patrick Gorman, thank you again) did appear to know both the Macintosh and the XVT product well. As an example, I asked for some help with implementing Apple events, and I was given good instructions.

A number of companies are "development partners" with XVT and offer extension products with compatible APIs, ranging from word-processing engines to connectivity/database

front-end extensions. Not all of these products are Mac-compatible. The guide clearly states which products are compatible with which platform. I have been told by the sales staff at XVT that a release of XVT DSC++ that is due out shortly will include a number of "Power Objects", which are complex interface objects that simplify GUI tasks. A Table object is included as part of the tutorial.

All executable code is in precompiled libraries, with the exception of XVT Power++, which comes also as source code. The price is $2325 per platform per developer; 68K and PPC Mac are considered two different platforms. A complete source code version is over $29,000. The primary clients appear to be large corporations doing in-house programming, or programmers contracting with a large corporation to do the same. The sums involved are substantial, and the relationship between client and vendor potentially a long one. XVT has a program whereby a prospective client gets special technical support with calls, emails, etc., while the product is being evaluated. I was told by the sales staff that a prospective purchaser can attend XVT's classroom program on using DSC++ in Colorado without cost (you supply airfare, room, board; they supply the training), to get a good feeling for how the product works and to see if it really is for your company. XVT has a clear sense that their best sales tool is making sure, as I put it earlier, that you "get it". I think they are right; a happy customer is a return customer, and probably makes tech support simpler.

Did I Get It?

I think so. XVT Software has produced a product that implements a C code interface to a common subset of the GUI features of a variety of platforms. It has produced a resource language that supports this GUI mechanism. It has created a quality class library and application framework that uses this interface to create platform-independent applications using C++. XVT Architect is a visual tool for creating class relationships and drawing the GUI. The C++ product is very much data-oriented, supporting out of the box a much wider array of data types than native Mac frameworks, and has a mechanism that simplifies sharing data among multiple views. There is a lack of finish in the tools, and the code itself is not as easy to use as I'd like. I was quite disappointed by the behavior of the example projects. The problems with the tools, the example code, the slow update of the CW libraries, and the difficulty with the FTP account left me with a lingering overall caution about the product despite underlying good documentation and thoughtful overall product design. The applications produced are large and slow, and, though the matter is hard to quantify (and XVT Software will probably disagree), there seems to be a general underlying incomprehension of or indifference to the Macintosh Way – the sense of style, the attention to detail, the ease of use (even for programming tools) that we take for granted. Finally, this product, despite its positives and the good intentions of XVT Software, is constrained to a common denominator defined by less sophisticated platforms.

I would not use this program to write my killer app and then port to Windows. However, efficiency, ease of use and productivity are ultimately in the eye of the beholder. A large contracting concern or a large corporation with an in-house programming department might well choose a product such as XVT, declare it the standard, and find that it meets the needs of the company and/or the contract more than adequately. Indeed, the efficiency of programming and meeting corporate MIS needs could more than make up for the speed bumps and lack of polish that would be encountered when a uniform solution was demanded of many individuals using multiple platforms. Thus, I could see XVT DSC++ as a potentially useful tool for a multiple-platform in-house programming staff. On the other hand, certain applications might suffer from the speed issues and the lack of ability to customize and take full advantage of the Mac interface. Ultimately, my recommendation is to take full advantage of XVT's evaluation program, and make your own decision.

SYMANTEC TOP TEN

By Craig Conner

SYMANTEC.™

This monthly column, written by Symantec's Technical Support Engineers, is intended to give our readers technical information on using Symantec products.

This month we start with a few THINK Pascal questions, then run through questions dealing with exception handling (or lack thereof), and then finish off with some miscellaneous topics.

The first question is from a news post:

Q: I have a large THINK Pascal project that I have always compiled for 68K with a floating-point unit. I now want to compile for a 68030-based PowerBook that has no FPU. Unfortunately, I am having some problems linking. If I turn off **68881/68882** under **Code Generation** in the **Compile Options** dialog, I get a link error:

```
undefined: 3SINH2 (Activation.p)
undefined: 3TANH2 (Activation.p)
```

I need to use these functions but I do not know how to get rid of this error. I did remove the SANELib881.lib library and replace it with SANELib.lib, but I still get the error. Any suggestions?

A: Unfortunately, those functions are defined in `SANE.p` only if you are compiling with **68881/68882** on. Fortunately, here are `sinh`, `cosh`, and `tanh` defined:

```
function SinH(x:real):extended;
begin
   sinH:=(exp(x) - exp(-x))/2;
end;

function CosH(x:real):extended;
begin
   cosH := (exp(x)+exp(-x))/2;
end;

function TanH(x:real):extended;
begin
   TanH := SinH(x)/CosH(x);
end;
```

TanH can be optimized like this:

```
function TanH(x:real):extended;
var
   eToX, eToNegX:extended;
begin
   eToX := exp(x);
   eToNegX := exp(-x);
   TanH := ( eToX - eToNegX ) / (eToX + eToNegX);
end;
```

It will be a tad slower than using SANE, but it will work. You can look at how `exp(x)` works on p. 358 of the *THINK Pascal Users Manual*, and the hyperbolic functions in any trigonometry textbook.

Q: When using THINK Pascal, I want to create my own pattern for use with the `PenPat` routine. How do I do this?

A: You should use the `StuffHex` routine (QuickDraw) to build your patterns. `StuffHex` will interpret its argument as an 8-by-8 bit-pattern with 1's denoting "on" bits, and 0's denoting "off" bits. Like so:

```
var
 myPat :Pattern;

StuffHex(@myPat, 'FFFFFFFFFFFFFFFF'); {All on}
```

Q: In THINK Pascal, whenever I try `TextFace(bold)`, I get an "incompatible type" error. Why?

A: The "styles" used in Pascal are values in a set and need to be expressed as `[bold]`, `[underline]`, `[italic]`, etc. Normal text is specified as the empty set, `[]`. Try:

```
TextFace([bold]);
```

Q: In a TCL project that I am updating to version 8.1, I get an "undefined" error on `catch_`. What might cause this?

HELP MAKE MACTECH WORK

Here at *MacTech Magazine*, we rely heavily on outside writers for most of the material that appears in our pages. If readers did not participate in the magazine, sending us their ideas and taking the time to write articles, there would be no *MacTech*. *MacTech Magazine* is not a staff of writers sending a constant stream of one-way messages outwards; it's a living, evolving network of readers conversing with one another, educating one another, sharing their knowledge, their experience, their interest, their trials and tribulations and joys and successes in the constantly unfolding story of programming the Macintosh. *MacTech Magazine* doesn't just happen: it's what the community makes it. If we carry reports of future trends and technologies, if we teach useful methods, if we review new books and tools, if we provoke thought, provide help, ride the wave of current interests and concerns, it is only because we reflect the thoughts of our readers, who speak through our pages.

You are invited to involve yourself in this exciting conversation amongst readers. No matter who you are, no matter what your credentials may be, if you have a tale to tell, a trick to share, a technique to teach, we want you to consider joining the family of those who write for *MacTech*.

Don't just wait for a topic to be covered or a technique explained in *MacTech!* Take responsibility! Write us an article yourself!

To write for *MacTech*, just send for our Writer's Kit. It's a Microsoft Word file containing the Styles you need to use, and giving lots of helpful advice and information, including all the legal stuff. You can let us know what you're writing about, or, if you want to, you can just write the article and spring it on us when it's done. [**Note:** We also have a need for people willing to make themselves available to write occasional product/book reviews.] If we publish your article, you'll be paid for it!

Write to us, the editorial staff, at editorial@mactech.com (or one of the other addresses listed on page 2 of the magazine). Take the future of *MacTech Magazine* into your own hands!

For Macintosh Programmers & Developers
MacTech™ MAGAZINE

A: If you are using native exceptions (which is standard for 8.1 projects), `catch_` has been redefined as `catch_reference`. Here is an example:

```
try_
{
   DoIt();
}
catch_reference(CException, e)
{
   errVal = e.GetErr();
}
catch_all_()
{
   errVal = -1;
}
end_try_
```

Q: I have just completed my latest project using TCL, and want to use MrC++ for the finishing touch. However, I get link errors when trying to build it. Why?

A: MrC and MrC++ do not have native exceptions built in, so you need to change the libraries some to get your project to link correctly. Remove the current CPlusLib TCL.o and BRLib libraries, and replace them with the ones that use the non-native exception handling. These are CPlusLib TCL_BELeh.o and BRLib(non-native eh).o. That should get it working.

Q: I just saw a note in the fp.h header recently about `dtox80` and `x80tod` being contained in a library called MathLib v.2. Why was it not included on the last CD?

A: That note in the header file is premature. Apple has not released a new version yet. Currently you can use `ldtox80` and `x80told` and they will work correctly.

Q: While compiling the OpenDoc project ODFFramewrk.RB.π I get "not enough memory" errors when trying to load the precompiled header called ODFHeaders.RB. How can I fix this?

A: The obvious answer is to get more RAM. OpenDoc will require at least 32 megs, and the more the merrier. Turning on Virtual Memory should also work, but will slow compiling down. If Virtual Memory is too slow, you can change the size of the precompiled header using a switch set up for the headers. In the ODF.pch file, there is a:

```
#define FW_AGGRESSIVE_PRECOMPILE 1
```

Change this to 0 and then do the same for FWEnvDev.h. After re-precompiling, the ODFHeaders.RB file will be smaller.

Q: I am trying to build a 68K version of one of my SPM projects. I get the following error at link time using Link via ToolServer:

```
### While reading file "long pathname:MPW68KRuntime.o.o"
### Link: Error: PC-relative edit, offset out of range.
    (Error 48) %__MAIN (309)
    Reference to: main in file: pathname:main.cp.o
```

A: There are some nuances that you need to take care of when using ToolServer to link a 68K application in the link order file (the .lo file). The MPW68KRuntime library uses Near

code, so it needs to be near the top of the list (in one of the first two segments that get created). Also, since that library calls the main routine, the file that contains that routine needs to be near that library. If you get a similar error after rearranging, then you need to turn on Far Code in the linker options, and use Far libraries where possible. Any library that does not have a Far version (like the runtime lib) needs to go near the top of the list.

Q: When using the `alloc_gla` memory package with the `Vector` demo, it reports heap corruption. What is wrong with the example?

A: Looking at the example, we see code like this:

```
vector(int size)    { v = new T[size]; sz = size; };
~vector()           { delete v; };
```

This code may have been correct at one time, but nowadays it is not. Since the `new` call allocates an array, the `delete` operator is no longer the proper operator to use. Instead, the `delete[]` operator should be used, and `alloc_gla` will stop complaining:

```
vector(int size)    { v = new T[size]; sz = size; };
~vector()           { delete[] v; };
```

Q: If I realign a structure like so:

```
#pragma options align=mac68k
struct{
...
}myStruct;
#pragma options align=powerpc
```

my program crashes when trying to use `myStruct`. What might be happening?

A: The default setup for PPC projects is to use four-byte boundaries. `align=powerpc` will align to two-byte boundaries, and could cause incompatibilities. Use `#pragma options align=reset` instead, and that should fix the problem.

PROGRAMMER'S CHALLENGE

By Bob Boonstra

BYTE CODE INTERPRETER

September is Assembly Language month at the Programmer's Challenge, and this year we will be accepting solutions in PowerPC Assembly for the first time. This month's Challenge, suggested by Xan Gregg, is to write an interpreter for a subset of the byte code language used by the Java Virtual Machine.

The prototype for the code you should write is:

```
void JavaMiniVM(
   void *constant_pool,    /* pointer to cp_info array */
   void *fields,           /* pointer to field_info array */
   void *methods,          /* pointer to method_info array */
   void *classFile,        /* pointer to class file */
   long methodToExecute,   /* index of method to start executing */
   void *heapSpace,        /* preallocated storage for your use */
   void *returnStack       /* stack where return values are stored */
);
```

Your Challenge is to write an efficient interpreter for a subset of the Java byte code instruction set. A Java instruction consists of a single-byte opcode specifying the operation to be performed, followed by zero or more operand bytes. So, for example, in the byte sequence `0x10 0xFF`, the opcode `0x10` (`bipush`) indicates that the operand byte `0xFF` is to be pushed onto the operand stack. The instruction `0x60` (`iadd`) indicates that two integers are to be popped off the operand stack, added, and the result pushed back onto the stack. The virtual machine operates by repeatedly fetching an opcode and performing the indicated action on the operands.

To participate in this Challenge, you don't need to know anything about Java itself, but you do need to understand the Java Virtual Machine. A Java Virtual Machine executes a .class file, the format of which is too complicated to provide here; it is described in the Java Virtual Machine Specification (release 1.0 Beta) available at http://java.sun.com/java.sun.com/newdocs.html.

The first three parameters passed to your `JavaMiniVM` routine are pointers to the constants, fields, and methods contained in the .class file that your interpreter is to execute. These parameters are taken directly from the `classFile` described in Section 2 of the VM specification. A pointer to the `classFile` is provided as the fourth parameter for those who feel they need direct access to the .class file. The parameter `methodToExecute` indicates which of the `methods` your VM is to start executing.

Stack space and execution frames should be established by

THE NEW DEADLINE

The Challenge is due on the **first** of the month, not the tenth, as has been the case since the column started. To get the Challenge problem in time, you should be subscribed to the Programmer's Challenge Announcement List.

Subscribe by sending an email to "listserv@listmail.xplain.com", with the **subject** "subscribe challenge-a".

There's also an open discussion list to discuss Challenges, review solutions, and talk about efficient computing. You subscribe to this list by sending an email to "listserv@listmail.xplain.com", with the **subject** "subscribe challenge-d".

And finally, there's an FTP site for members of the Challenge lists. Its URL is:
ftp://ftp.mactech.com/mactech/CHALLENGE.

People have used the FTP site to trade sample data and upload alternative solutions.

THE RULES

Here's how it works: Each month we present a new programming challenge. First, write some code that solves the challenge. Second, optimize your code (a lot). Then, submit your solution to *MacTech Magazine*. We choose a winner based on code correctness, speed, size, and elegance (in that order of importance) as well as the submission date. In the event of multiple equally desirable solutions, we'll choose one winner (with honorable mention, but no prize, given to the runner up). The prize for each month's best solution is a $100 credit for Developer Depot™ and a limited-edition "The Winner! MacTech Programmers Challenge" T-shirt (not available in stores anywhere).

Unless stated otherwise in the problem statement, the following rules apply: All solutions must be in ANSI compatible C. Use only pure C code. We disqualify entries with any assembly in them (except for challenges specifically stating otherwise). You may call any Macintosh Toolbox routine (e.g., it doesn't matter if you use NewPtr instead of malloc). We test entries with compiler options set to disable FPU use (for 680x0 code) and to enable all available speed optimizations. The compiler to be used and the target instruction set (680x0 or PowerPC) will be stated in the problem. **Limit your code to 60 characters per line;** this helps with e-mail gateways and page layout.

We publish the solution and winners for each month's Programmer's Challenge two months later. All submissions must be ***received by*** the 1st day of the month printed on the front cover of this issue.

You can get a head start on the Challenge by reading the Programmer's Challenge mailing list. It will be posted to the list on or before the 12th of the preceding month. To join, send an email to listserv@listmail.xplain.com with the message "subscribe challenge-A".

Mark solutions "Attn: Programmer's Challenge Solution" and send it by e-mail to one of the Programmer's Challenge addresses in the "How to Communicate With Us" section on page 2 of this issue. Include the solution, all related files, and your contact info.

MacTech Magazine reserves the right to publish any solution entered in the Programmer's Challenge. Authors grant *MacTech Magazine* the exclusive right to publish entries without limitation upon submission of each entry. Authors retain copyrights for the code.

WHY NOT USE THE BEST?

Companies like Adobe, Claris, Symantec, Metrowerks, Netscape and hundreds of others recognize the unsurpassed quality of Developer VISE and have switched to MindVision for their installation needs. In fact, every day MindVision sells more installers than all other competitors combined. Don't take our word for it. Ask any of our hundreds of satisfied customers why they switched from the competition to MindVision's Developer VISE.

We didn't stop there.
Announcing...

INSTALLER VISE 4.0

The New Industry Standard for Software Installers.

Find, move, rename, or delete any file. Create hierarchical packages. Build one installer for multiple languages. Build installers with unparalleled ease. Attach AppleScripts to your installation. New archive features allow you to build CD-ROM installers quickly and easily. It may not slice and dice, but in addition to installing it also removes. And much, much more.

And Introducing...

UPDATER VISE 1.1

The Most Advanced Updater Available.

Updater VISE is the only product that can update multiple versions of your 68K, PowerPC, and Fat application from ONE updater. Updaters are extremely compact and totally secure, allowing for easy online distribution. And, combined with Installer VISE you can update any number of different files giving you a total software delivery solution.

Maybe it's time to stop and take a look at MindVision's software delivery solutions.
Visit our web site for full information,
including fully-functional demos.

MindVision Software

www.mindvision.com

Email: info@mindvision.com • Voice: (402) 477-3269 • Fax: (402) 477-1395

VISE SOFTWARE DELIVERY SYSTEM

your virtual machine in the memory provided in `heapSpace`. Adequate heap space will be allocated by the caller. Your code may include static data that might be needed for lookup tables, etc., to efficiently implement the virtual machine. The parameter `returnStack` is provided as the stack for the execution environment of the calling routine. It is to be used when executing the various `return` byte codes to provide the caller access to your results.

To simplify the Challenge, your code need not implement the `long`, `float`, and `double` data types supported by the Java Virtual Machine. You also do not need to process exceptions, breakpoints, monitored code regions, or the `wide` modifier for Load and Store instructions. Your interpreter should be robust enough to determine the operand size of these unimplemented instructions in order to skip any that are encountered. All methods invoked will be in the single **.class** file provided to your routine.

Sample test **.class** files will be provided via the Programmer's Challenge mailing list, and are also available by writing me at bob_boonstra@mactech.com. If there are any questions about what needs to be implemented, please send me a note at the same email address.

Your code may be written in PowerPC Assembly, 68K Assembly, C, or C++. Testing will be performed on an 8500 using the latest CodeWarrior environment. Because this is a more difficult Challenge than usual, it has been sent to the mailing list earlier than normal to provide additional solution time. For those of you who haven't had a chance to investigate Java in detail, it is a good opportunity to find out what all the excitement (hype?) is about. I hope you find this Challenge enjoyable and educational.

Two Months Ago Winner

Once again, congratulations go to **Ernst Munter** (Kanata, Ontario), this time for submitting the fastest entry to the Connect IV Challenge. Recall that the Challenge was to compete in a round-robin tournament against the other solutions in a generalized version of the well-known Connect 4 game. Pieces are inserted into the top of a column in the vertically oriented board, with the winner being the first player to arrange four or more pieces into a vertical, horizontal, or diagonal line.

Of the five entries I received, four worked completely or nearly correctly, although two of these occasionally forfeited a game by making an illegal move or incorrectly claiming victory. The two solutions winning the most tournament points required more execution time than the others, with the winning solution requiring the greatest amount of time for all but the largest board sizes. The winning solution uses a data structure dubbed a `quad` to denote each possible line of four pieces passing through a given point on the board, and keeps track of whether each `quad` represents a possible win for either player. Comments in the solution describe heuristics used to prune the recursive search for the best move.

Four test cases were used, consisting of different board sizes. Each solution competed twice against each other solution, playing once with the first move and once with the second move. The table below indicates how many points were earned by each entry for each of the board sizes:

Name	7x7	16x13	33x48	63x62	Total Points
Greg Cooper	6	6	6	8	26
Louis Deaett	0	4	2	4	10
Ernst Munter	12	10	12	12	46
Keith Pomakis	6	4	4	0	14

The table below summarizes the results for each entry, including tournament points, code size, data size, and execution time. Numbers in parentheses after a person's name indicate that person's cumulative point total for all previous Challenges, not including this one.

Name	points	time	size
Ernst Munter (194)	46	6212	2944
Greg Cooper (7)	26	7272	1600
Keith Pomakis	14	111	3512
Louis Deaett	10	<1	4600

Top Twenty Contestants

Here are the top twenty contestants for the Programmer's Challenge. The numbers below include points awarded over the twenty-four most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	203	11.	Cutts, Kevin	21
2.	Gregg, Xan	92	12.	Picao, Miguel Cruz	21
3.	Larsson, Gustav	87	13.	Brown, Jorg	20
4.	[Name deleted]	67	14.	Gundrum, Eric	20
5.	Lengyel, Eric	40	15.	Karsh, Bill	19
6.	Lewis, Peter	30	16.	Stenger, Allen	19
7.	Darrah, Dave	29	17.	Cooper, Greg	17
8.	Beith, Gary	24	18.	Mallett, Jeff	17
9.	Kasparian, Raffi	22	19.	Nevard, John	17
10.	Vineyard, Jeremy	22	20.	Nicolle, Ludovic	14

There are three ways to earn points: (1) scoring in the top five of any Challenge; (2) being the first person to find a bug in a published winning solution; or (3) being the first person to suggest a Challenge that I use. The points you can win are:

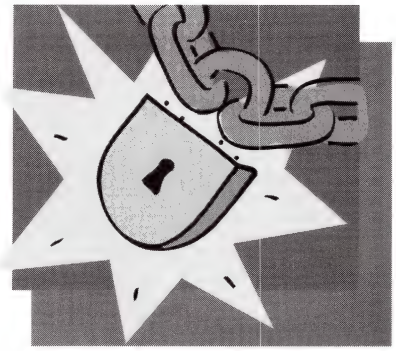
1st place .....20 points
2nd place.....10 points
3rd place .....7 points
4th place.....4 points
5th place.....2 points
finding bug.....2 points
suggesting Challenge...2 points

dtF The Relational Database System

dtF v1.6 including numerous SQL enhancements and ODBC driver available!

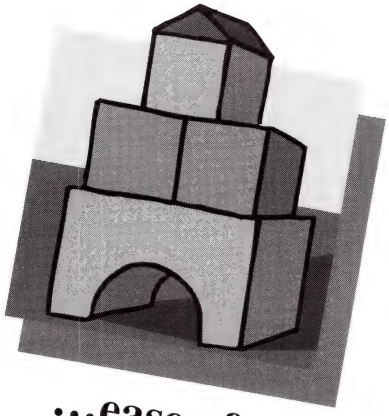
When it comes to performance **dtF** is in a class all its own. **dtF** utilizes a proprietary query optimization and caching scheme to obtain unparalleled performance. If you are in need of a quantum leap in performance, use the **dtF** native PowerPC stand-alone or client/server version.

...performance



Full transaction control and automatic level-3 error recovery guarantee maximum data protection even after sudden system crashes. **dtF** databases are compressed and encrypted to protect against all unauthorized access, even disk editors. Implement sophisticated table and operation level security with **dtF**'s password features.

...data security



...ease of use

With **dtF**'s unique direct integration technology, the **dtF** database system is fully contained in your application, enabling you to build double-clickable database applications. Never worry about missing or conflicting INITs or drivers. **dtF**'s native API is compact, easy-to-use and integrates seamlessly with all major development environments on the Macintosh. **dtF**'s high performance SQL, integrated data dictionary, security features, automatic index selection, query optimization and error recovery allow you to concentrate on creating great database applications instead of messing around with the internals of the database system.

Stand-alone applications built with dtF are royalty-free!

dtF is available for Macintosh System 7.x (68K and native PowerPC). **dtF** supports MPW C/C++, Symantec C/C++, Metrowerks CodeWarrior (all compilers 68K and native PPC). **dtF** is fully OpenDoc™ compatible. Separate versions for use with HyperCard 2.x, SuperCard 2.x, Smalltalk-Agents and Pictorius Peregrine are provided. AppleScript interface via DataScript™ for **dtF** from General Knowledge. **dtF** supports cross-platform development on Windows 3.11, Windows 95 and OS/2.

dtF Americas, Inc.
19672 Stevens Creek Blvd.,
Suite 128
Cupertino, CA 95014
USA

Phone: (800) DTF-1790
Fax: (510) 828-8755
AppleLink: DTF.AMERICA
Internet: dtF.america@applelink.apple.com

dtF The Relational Database System

Here is Ernst's winning solution:

CONNECTMOVE.CP

Ernst Munter

```
/*
```

"Connect IV"

The challenge is to write code for a well-known game that will compete against other entries, as well as against the clock. The game board consists of an NxM array into which two players alternate inserting pieces. Pieces are inserted into the top of a column in the vertically oriented board, so that they drop into the lowest unoccupied cell of that column. The objective is to be the first player to arrange 4 or more pieces into a vertical, horizontal, or diagonal line.

Solution

My solution is based on an unsophisticated look-ahead scheme. We search, depth first, up to a maximum depth, to find the move which will give the highest board score. Dummy moves are executed alternatingly by the 2 players, in a recursive function.

At each recursion level, the board score accumulated at the deeper levels is subtracted from the value of the move contemplated at this level.

In a full look-ahead to depth N, and with a board width of C columns, we would have C^N moves to examine.

Two techniques are combined to reduce the number of moves to be examined:

At each level, the search can stop if a certain threshold is exceeded or a winning (losing) position is reached. Exceeding the threshold would mean that the calling level could not gain a better score, given its best score to this point.

In order to be able to stop searching (prune the tree), we would like to start with moves which give the better scores, i.e. rather than go from left to right across the board, we start at the position of the last move and go alternatingly left and right of that point. This tackles the active area of the board first where it is more likely to find the best move.

Data Representation

The scoring relies entirely on a simple representation of the state of the board.

The board is visualized as covered with potential lines (horizontal, vertical, diagonal) of 4 adjacent fields. Each such line is called a "quad".

A given field is associated with at least 3 (corner fields) and at most 16 such quads (in the center of a large board).

Each quad can have a state (empty, owned by player 1, owned by player 2, or spoiled). The spoiled state implies this quad contains pieces from both players, and can no longer be a winning quad.

In addition, a non-empty quad has a value, depending on how many player pieces it contains.

I have arbitrarily set these values as follows:

empty	0
1 piece	1
2 pieces	16
3 pieces	256
4 pieces	4096

The value of a placing a piece in a given field, is calculated as the sum of the values of all quads associated with this field.

To simplify the book keeping, each field has an associated data structure which points to all relevant quads.

The Field structure also provides local stack space to hold the previous values of its quads while a move is explored.

A recognized weakness of this approach is that it treats all quads equally, and does not recognize gravity, that is the need to fill columns from the bottom. This effect can shadow otherwise good candidates from ever winning. I had no time left to include this consideration in the board scoring.

Running Time

The depth of the tree dramatically affects the time to compute a move, as does the width of the board.

A few minor techniques are used to improve running time.

After each move, all spoiled quads are removed from their field references.

All full columns are excluded from the move list.

Empty columns "far away" from the action are also excluded.

The MAGIC_NR constant can be used to adjust the speed of the program. A higher value increases the search depth and the running time.

I set MAGIC_NR to 17 for acceptable all-round performance on my computer.

Assumptions

At least 7 columns are assumed.
The calling program should check for wins;
it is assumed that this function is not called
if the opponent has already won.

A move value of -1 is returned when errors are detected, e.g when there are no free fields left.

This program does not include a randomizer since it is assumed it will play only a few times against computer opponents who, we hope will not be able to take advantage of this.

In a version to be played by humans, one would randomly choose between equally good moves to make it more interesting.

Note on style

This program is basically a C program in spirit, but using C++ structs for convenience in accessing dynamic data.

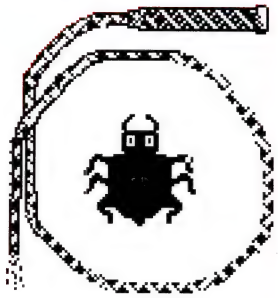
No inheritance, operator overloading or the like.

Gives you the Information to *Program your Best!*

The Debugger V2 & MacNosy

by Steve Jasik

Information



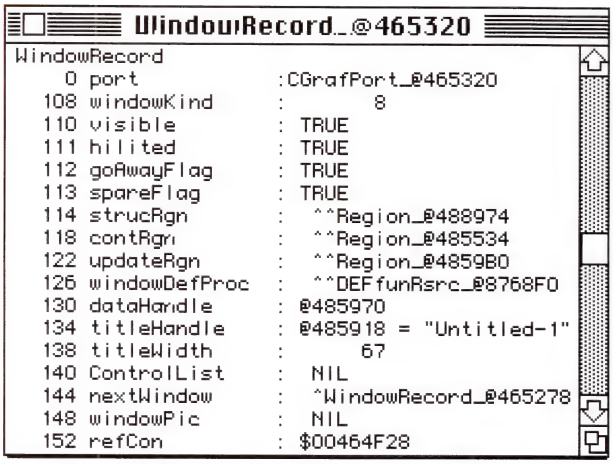
Control

For over 10 years Steve Jasik has brought Macintosh Developers the superior debugger. In addition to the software, Steve provides timely support to his customers. Inspect the feature list and see why so many developers use it. In addition to basic features such as stepping and breakpoints, the Jasik debugger has many advanced features such as heap display, a fast memory watch, Soft-MMU, jump tracing, ... to help you track down the elusive bugs that other debuggers won't help you find.

In additon, the package includes **MacNosy,** a global interactive disassembler that enables one to recover the source code of any Mac application or from the ROM.

When you compare features of the different debuggers, note that *only one* has all the below features to help you get your job done, and *only one* has MacNosy to help you debug any program in a full system environment symbolically!

"The Debugger" is the debugger of choice at: Adobe, Apple, Claris, Kodak, Metrowerks, etc.



An example of a structured data display window with hypertext links to substructures

Its Features Include:

- **Source level debugging for Metrowerks & MPW** compiled programs (C++, C, Pascal, Fortran, ...).
- **Symbolic Debugging** of *any* Macintosh program, ROM, Plug-In or *code resource* (DRVRs, XCMDs, INITs, PDEFs..), ASLM & **CFM LIbraries, OpenDoc parts**, GX Drivers, ...
- **Simultaneous Symbolic debugging of multiple "tasks"**
- **Object Inspector for MacApp 3 & PowerPlant programs**
- **Source level debugging of Symantec C™ projects**
- **Includes a program (CoverTest) to interactively do Code Coverage analysis for SQA testing, etc.**
- **Fast Software Watchpoint** to find clobbered variables.
- **Soft MMU** ("heap centric" bounds checking) for PowerPC programs to find out of bounds references.
- **Jump Trace** to find where a program takes a "wild jump"
- **Sophisticated error check algorithms** such as Trap Discipline (Argument Checking), Heap Scramble and Heap Check to detect errors before they become disasters
- **Structured display** of data (hypertext) with user definable structures while debugging
- **Conditional breakpoints** to filter redundant information
- **Continuous Animated Step Mode** to watch your program execute instruction by instruction
- **Detailed symbolic disassembly for both 680x0 and PowerPC** with symbol names, labels, cross ref maps, – make it possible to ferret out the secrets of the ROM, etc.
- **"Training Wheels"** for the PowerPC disassembler to help you learn the opcodes

The Debugger V2 & MacNosy: $350

Runs on all Macs. Call For Group prices or Updates.
Visa/MC Accepted.

Available from: Jasik, APDA or ComputerWare (800-326-0092).

Jasik Designs • 343 Trenton Way, Menlo Park, California 94025 • (415) 322-1386

URL: http://www.jasik.com e-mail: macnosy@jasik.com

All simple functions are listed inline, as part of the class. The implementations of functions with loops are listed separately, following the struct declaration.
*/

```
#include <stdlib.h>
#include "viervier.h"

#define maxRow 63
#define maxCol 64

#define maxQuad (((maxRow-3)*maxCol)+      \
  ((maxCol-3)*maxRow)+(maxRow-3)*(maxCol-3)*2)
#define empty     0
#define self      1
#define opponent  2
#define spoiled   3

#define OTHER_PLAYER (3-player)
#define WIN  4096
#define MAX_MOVES 9

#define MAGIC_NR  17
```

Quad

```
struct Quad {
  short  status;      //who owns quad
  short  value;       //16 ^ (n-1);

  int Update(int currentPlayer){
    if (status==empty) {
      status=(short)currentPlayer;
      value=1;
      return value;
    }
    if (status==currentPlayer) {
      value <<= 4;                //higher value
      return value;
    }
    if (status != spoiled) {      //other player
      status=spoiled;
      return value;                     //plus for us
    }
    return 0;                           //already spoiled
  }
};
typedef struct Quad Quad;
```

Field

```
struct Field {
  Quad* quadRef[16];              //pointers to intersecting quads
  Quad savedState[16];            //stack to save quads before move

  void AddQuad(Quad* qp);
  void SaveState();
  void RestoreState();
  int MakeMove(int currentPlayer);
  void Rationalize();
  int IsEmpty(){
    if (quadRef[0]->status) return 0;
    return 1;
  }
};
typedef struct Field Field;
```

Field::AddQuad

```
void Field::AddQuad(Quad* qp) {   //adds qp to list of quads
int k;
  for (k=0;k<16;k++) {
    if (0==quadRef[k]) {
      quadRef[k]=qp;
      break;
    }
  }
}
```

Field::SaveState

```
void Field::SaveState() {         //save all quads on stack
Quad*  qp;
Quad** qh=quadRef;
Quad*  savePtr=savedState;
int  k;
  for (k=0;k<16;k++) {
    if (0 == (qp=*qh++)) break;
    *savePtr++=*qp;
  }
}
```

Field::RestoreState

```
void Field::RestoreState() {      //restore quads from stack
Quad*  qp;
Quad** qh=quadRef;
Quad*  savePtr=savedState;
int  k;
  for (k=0;k<16;k++) {
    if (0 == (qp=*qh++)) break;
    *qp=*savePtr++;
  }
}
```

Field::MakeMove

```
int Field::MakeMove(int currentPlayer) {
long sum=0;
long val;
Quad*  qp;
Quad** qh=quadRef;                //move=update all quads
                                  //return value of move
int  k;
  for (k=0;k<16;k++) {
    if (0 == (qp=*qh++)) break;
    val=qp->Update(currentPlayer);
    if (val>=WIN) {
      return WIN;
    }
    sum+=val;
  }
  return sum;
}
```

Field::Rationalize

```
void Field::Rationalize() {
int  i=0;
int  k;
int  nq=0;                  //remove all spoiled quads

//find number of non-0 quad refs
  for (k=0;k<16;k++) {
    if (quadRef[nq]) nq++;
    else break;
  }

//scan quad refs for spoiled quads, and remove
  while(i<nq) {
    Quad* qp=quadRef[i];
    if (qp->status==spoiled) {
      nq--;
      if (i<nq) quadRef[i]=quadRef[nq];
      quadRef[nq]=0;
    }
    i++;
  }
}
```

PrivateData

```
struct PrivateData {
  int nextMove;         //next real move to do
  int numCols;          //number of columns
  int numRows;          //number of rows
  int numFree;          //number of fields free
  int level;            //recursion depth
  int numMoves;         //number of moves in move list
  Field* endOfBoard;    //sentinel pointer
  int numHistory;       //move counter
  int history[maxCol*maxRow];  //move history
  int moveList[maxCol];        //list of columns
  Field* nextField[maxCol];    //next free field in column
  Field  board[maxCol*maxRow]; //array of all fields
  Quad quadSet[maxQuad];       //array of all quads

  void Initialize(long nCols,long nRows);
  void FirstMove() {nextMove = numCols >> 1;}
  void MakeMoveList(int move);
  int BestMove(int level,int player,int threshold);
```

```
  void Rationalize(int move);
  int  CannedMove();
  void RecordMove(int move) {
    if (numHistory==0)
      history[numHistory++]=move;
    else history[numHistory++]=move-history[0];
  }
  void AdjustLevel() {
    level=MAGIC_NR - numMoves;
    if (numMoves<=7) level--;
    if (level>=numFree) level=numFree-1;
  }
  int UpdateBoard(int move,int player) {
    Field* fp=nextField[move];
    long score=fp->MakeMove(player);
    nextField[move]=fp+numCols;
    Rationalize(move);
    numFree--;
    return score;
  }
};
typedef struct PrivateData PrivateData;

/*
field numbering example (6-by-7)

35  36  37  38  39  40  41   5  |
                               |
28  29  30  31  32  33  34   4  |
                               r
21  22  23  24  25  26  27   3  o
                               w
14  15  16  17  18  19  20   2  s
                               |
7   8   9   10  11  12  13   1  |
                               |
0   1   2   3   4   5   6    0  |
            - columns -
*/
```

PrivateData::Initialize

```
void PrivateData::Initialize(long nCols,long nRows){
int    col,row;
Field*   field;
Quad*    qp;

  numCols=nCols;
  numRows=nRows;
  numFree = nCols*nRows;

  endOfBoard=board+numFree;

  field=board;
  for (col=0;col<nCols;col++)
  nextField[col]=field++;

  field=board;
  qp=quadSet;
  for (row=0;row<nRows;row++) {   //set up quad references
    for (col=0;col<nCols;col++) {
      int direction,delta;
      for (direction=0;direction<4;direction++) {
           //right, up-right, up, up-left

  switch (direction) {
           //eliminate all that don't fit
case 0: if (col>=nCols-3) continue;break;
case 1: if (col>=nCols-3) continue;
case 2: if (row>=nRows-3) continue;break;
case 3: if ((row>=nRows-3) || (col<3)) continue;
  }

  for (delta=0;delta<4;delta++) {
    Field* fp;
    switch (direction) {
case 0: fp=field+row*nCols+(col+delta);break;
case 1: fp=field+(row+delta)*nCols+(col+delta);break;
case 2: fp=field+(row+delta)*nCols+col;break;
case 3: fp=field+(row+delta)*numCols+(col-delta);break;
```

```
      }
      fp->AddQuad(qp);
    }
    qp++;
      }
    }
  }
}
```

PrivateData::Rationalize

```
void PrivateData::Rationalize(int move) {
```

//remove spoiled quads from all free fields in the vicinity of a recent move

```
int i=move-3,j=move+4;
  if (i<0) i=0;
  if (j>numCols) j=numCols;
  for (;i<j;i++) {
    Field* fp=nextField[i];
    while (fp<endOfBoard) {
      fp->Rationalize();
      fp+=numCols;
    }
  }
}
```

PrivateData::MakeMoveList

```
void PrivateData::MakeMoveList(int move) {
int  m=move;
int  n=0;
int  k=0;
```

//build the move sequence, starting from the center
//out to the limits while skipping full columns.

```
  do {
    if (nextField[m]<endOfBoard) moveList[n++]=m;
    if (n>=MAX_MOVES) goto done;
    k++;
    if (0>(m=m-k)) goto go_right;
    if (nextField[m]<endOfBoard) moveList[n++]=m;
    if (n>=MAX_MOVES) goto done;
    k++;
    if (numCols<=(m=m+k)) goto go_left;
  } while(1);

go_left:
  if (0>(m=m-k-1)) goto done;
  do {
    if (nextField[m]<endOfBoard) moveList[n++]=m;
    if (n>=MAX_MOVES) goto done;
    m--;
  } while(m>=0);
  goto done;

go_right:
  if (numCols<=(m=m+k+1)) goto done;
  do {
    if (nextField[m]<endOfBoard) moveList[n++]=m;
    if (n>=MAX_MOVES) goto done;
    m++;
  } while(m<numCols);

done:
  numMoves=n;
}
```

PrivateData::CannedMove

```
int PrivateData::CannedMove() {
#define H0 (history[0])
  switch (numHistory) {
case 1: if (H0>=3) return H0;
  if (H0+3<numCols) return H0;
  return numCols/2;
case 2: switch (history[1]) {
  case 0:return H0;
  case 1:if (H0>=3) return H0-3; else return H0;
  case -1:if (numCols-H0>3) return H0+3; else return H0;
  case 2:case -2:return H0;
  case 3:if (H0>=1) return H0-1; else return H0-1;
  case -3:if (numCols-H0>1) return H0+1; else return H0+1;
```

```
   default:if (H0>=2) return H0-1; else return H0+3;
   }
case 3: if (history[1]==0) {
           if (history[2]>=0) return H0-1;
           if (history[2]<0) {
             if (H0+1<numCols) return H0+1;
           }
         }
   }
   return -1;
}
```

PrivateData::BestMove

```
int PrivateData::BestMove(
                int level,int player,int threshold) {
int i;
int bestMove=-1;
int moveValue;
int score;
int bestV=-0x4000L;

//this routine is called recursively to return the value of the best move. The class
//variable "nextMove" holds the column number for the "best" move.

   for (i=0;i<numMoves;i++) {
     int m=moveList[i];              //work from list
     Field* fp=nextField[m];
     if (fp < endOfBoard) {          //else column is full
       fp->SaveState();
       nextField[m]+=numCols;
       moveValue = fp->MakeMove(player);

       if (moveValue>=WIN) {         //win here: return right away
         nextField[m]=fp;
         fp->RestoreState();
         nextMove=m;
         return WIN*2;
       }

       if (level) {                  //descend to next level

   score=BestMove(level-1,OTHER_PLAYER,moveValue-bestV);
   if (score>=WIN) {
     moveValue=-score+1024;          //bias for depth
     goto skip;
   }
   moveValue -= score;               //and accumulate score
       }

       if (moveValue>=threshold) {
                                     //no need to search further
         nextField[m]=fp;
         fp->RestoreState();
         nextMove=m;
         return moveValue;
       }
skip:
       if (moveValue>bestV) {        //keep track of best so far
         bestV=moveValue;
         bestMove=m;
       }

       nextField[m]=fp;
       fp->RestoreState();
     }
   }
   nextMove=bestMove;
   return bestV;
}
```

ConnectMove

```
long ConnectMove (
   long numCols,
   long numRows,
   void *privStorage,
   long prevMove,
   Boolean newGame,
   Boolean *victory) {
```



```
PrivateData* PD=(PrivateData*)privStorage;

   if (newGame) {                    //initialize on first call
     PD->Initialize(numCols,numRows);
     if (prevMove==-1) {
       PD->FirstMove();              //standard first move
       goto finish;
     }
   }


   PD->UpdateBoard(prevMove,2);
   PD->RecordMove(prevMove);

   if (PD->numFree<=0) return -1;    //the board is full

   if ((PD->nextMove=PD->CannedMove())<0) {

     PD->MakeMoveList(prevMove);

     PD->AdjustLevel();

     PD->BestMove(PD->level,1,WIN);
   }
finish:
   *victory=(WIN<=PD->UpdateBoard(PD->nextMove,1));
   PD->RecordMove(PD->nextMove);
   return PD->nextMove;
}
```

By Rick Gansler

Driving Lotus Notes From an Application

Find API-ness in the land of the Lotus-Eaters

Lotus Notes is much more than a groupware or email program; it is a client/server document-oriented groupware database with a powerful API library that can be used to create standalone platform-independent C programs. With the introduction of Notes Release 4 for Mac OS 68K and PowerPC, Lotus has created a Mac client application that improves on Release 3 in innumerable ways. One such improvement is the inclusion of the Notes C API for Mac OS.

This article discusses the Notes C API and its implementation for the Mac OS. A sample API program is listed and explained. The sample program demonstrates how to export a Notes database in a form that can be compiled as a Newton Book for the Apple Newton PDA. Also discussed are the Notes C++ API and Vendor Independent Messaging (VIM) SDK, which will soon be available for the Mac OS.

Introducing Lotus Notes

Lotus Notes enables teams of people to communicate with each other, collaborate on shared documents, and generate custom workflow applications. The teams may all be connected to the same local area network, or may be connected via modem or remote access bridges.

Information is generally stored on a scaleable Notes server in the form of an encrypted relational database, in which various types of data can be stored. Users can create their own forms, providing a customized view of the data. Fax, voice mail, and pager gateways enhance the portability of the data. Interfaces to the Internet and legacy databases on mainframes further increase the scope of data portability.

The end user sees Notes as a window called the *workspace* (Figure 1) containing icons that represent databases. These databases may be stored on a Notes server or on a local disk or file server. Opening a database presents the user with a *view* (Figure 2) of the database, and a list of *documents* (Figure 3). A view may be thought of as the sorted result of a search. Each database may have one or more views, and always has a default view. Opening a document displays a *form* through which the document's *fields* (or *items*) are presented.

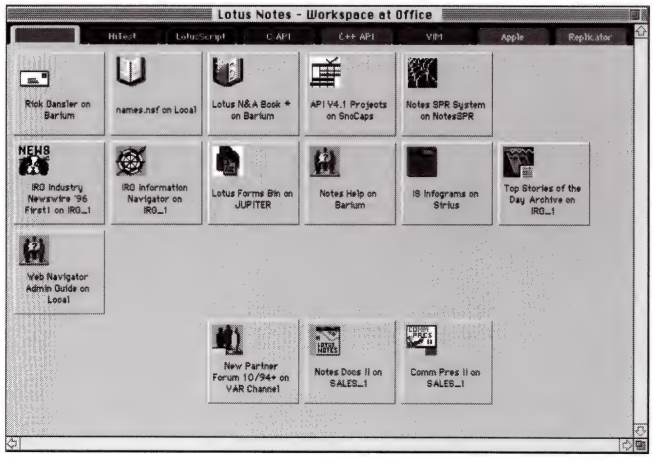


Figure 1. Notes client application workspace

Rick Gansler is a contractor working at Lotus on the Mac OS Notes C API Toolkit. Rick's responsibilities include developing strategies for integrating Apple-centric technologies (such as Newton) into the Notes API Toolkit. Rick first started doing Mac development on a Fat Mac in 1986, and has worked on many diverse projects since. He can be contacted at gansler@boardwalk.tiac.net; you can also check out his home page at http://www.tiac.net/users/gansler. For information about joining the Lotus developer program, please refer to http://www.lotus.com/partners/.

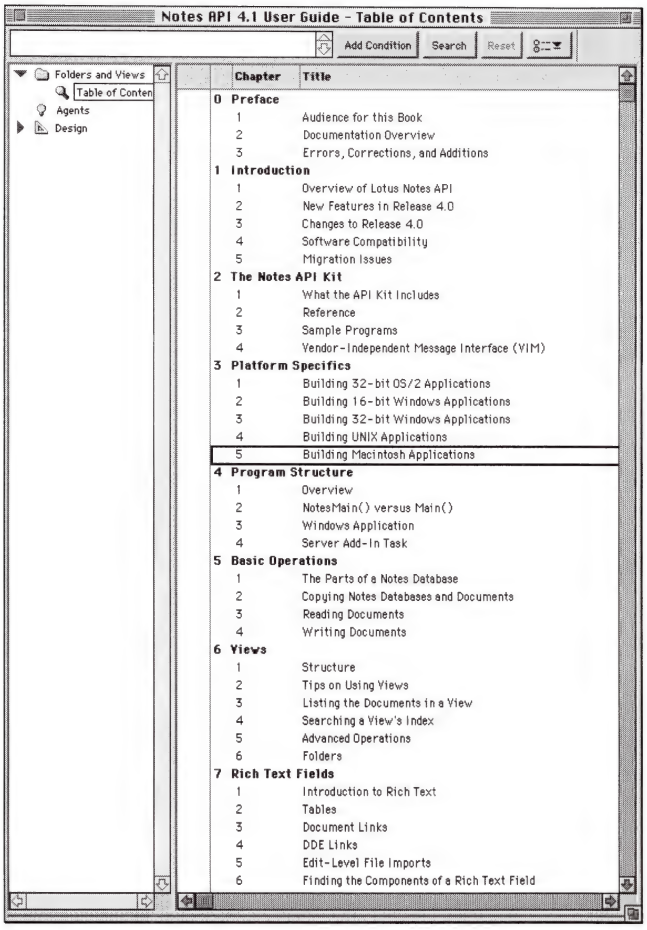


Figure 2. Notes client application database view

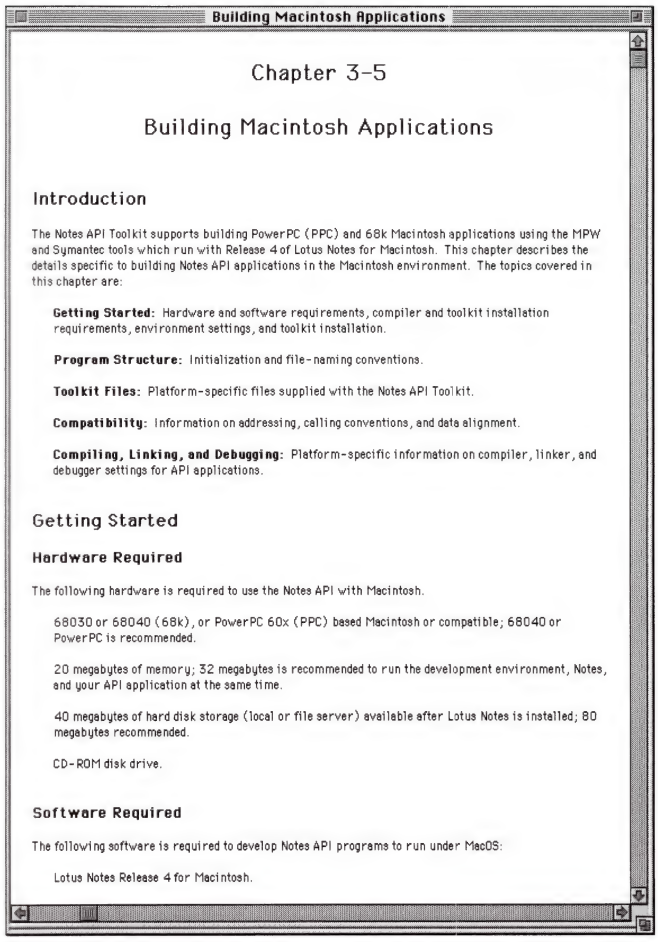


Figure 3. Notes client application document

What's So Special About Notes?

Notes shows its uniqueness in four areas: database replication, database design, cross-platform support, and application development.

Replication resembles synchronization of files, or mirroring of Web and FTP sites, only more powerful. Replication is bi-directional, and supports update, insertion, and deletion. It enables the same database to be stored on multiple servers and then updated so that all databases have the same contents, structure, and access controls. Databases can optionally be selectively replicated; this can be applied at the document or even at the field level. Field-level replication is especially useful when replicating over a modem, since less data needs to be transferred than when doing document-level replication. Having databases replicated across multiple servers is useful, since those servers can be distributed across organizations or time zones, which distributes the load of user activity among the different servers. Notes maintains a replication history, keeping track of when data was updated, as well as what the source of that data was, thus speeding future replications.

Using data access hooks into Notes, a Notes database can be synchronized with a legacy database. This feature enables an IS department to keep data and some applications on a mainframe (for example), yet still allow Notes users to access and modify that data from a desktop computer.

Databases can be designed using *templates* and custom *forms*, a number of runtime *views*, *@functions* (which take arguments, perform logical actions on data, and return a result), *Navigators* (which add a GUI with buttons and links on top of the database), *Agents* (scheduled and event-driven macros), *HotSpots* (like buttons), *DocLinks* (referring to any Notes document in any Notes database), and *URLs* (referring to a Web page via the InterNotes server).

Lotus Notes is truly cross-plaform, and is supported on Mac OS (client only), Windows 16-bit (client only), Windows 32-bit, OS/2 Warp, NetWare (server only), and many UNIX platforms including Solaris, HP/UX, and AIX. The Dec Alpha platform running Windows NT and Dec UNIX will soon be added to this list. A Notes client can access any Notes server, regardless of platform, using a variety of network protocols. For the most part, Notes for all platforms have identical functionality and an identical user interface.

Notes Application Development

There are two ways to accomplish Notes application development. The first is to use LotusScript, a cross-platform object-oriented event-driven interpreted language, which can be used in Notes as well as in other Lotus products. LotusScript is interpreted and cannot be run standalone. The second method is to create standalone programs using the Notes C API (or any of the other APIs, such as Notes C++ API, that are built on top of the C API); such standalones run independently of the Notes application, yet have access to Notes services. Just as different programming languages exist, each designed for a particular niche, Notes has multiple development options for programmers having different levels of experience and different development needs. Since the C API for Mac OS has only very recently been released, not all of the Notes API Toolkit development environments are available for Macintosh. Figure 4 shows the different environments, their level of object orientation, and their target audience.

Visual Basic Extensions (VBX) and OLE2 Extensions (OCX) are not available for Mac OS, so their Notes implementations cannot be made available either. The HiTest C API is an object-based (but not C++) layer built on top of the C API. Lotus has not yet determined if HiTest will be ported to Mac OS. Without HiTest on Mac, there can be no HiTest Glue, which is implemented as a VBX to provide Visual Basic programmers with controls that interact with Notes.

(Note that the C API can also be used to build code that is executed by Notes, for example to make import/export filters, menu add-in modules, and server add-in tasks. The subject, however, is beyond the scope of this article.)

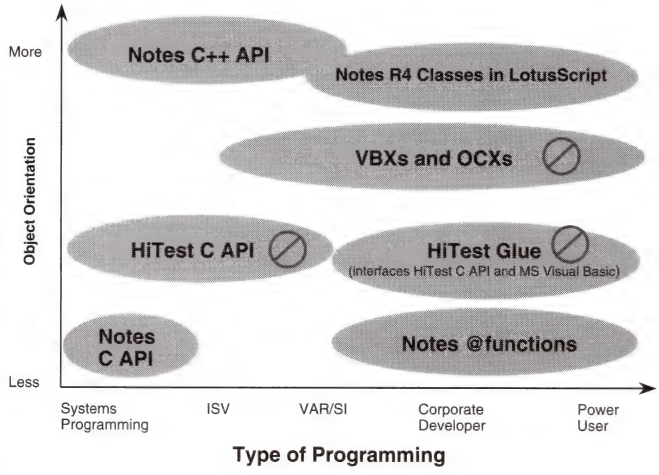


Figure 4. Notes programming methods

Notes Architecture

The Notes C API *is* Notes. The API is a large set of Notes functions that have been exposed for third-party developers to use. This means that the API is as important to Lotus as it is to developers; Lotus Notes is itself written using many of the same API calls that are available in the Notes API.

Admittedly, some parts of the Notes client and server applications use functions that are not available in the API; so an API programmer cannot write a program that does everything that the Notes client does. Recall, for example, the workspace window (Figure 1). This window's list of user databases and their icons is stored in a database called desktop.nsf; but Notes API calls cannot access this list.

Just the other way, though, some tasks available through the API cannot be accomplished through the Notes client application's user interface. For example, if a user has a document but doesn't have the form with which the document was created, it may not be possible to view all fields in that document using the Notes client. (A form defines what fields can be displayed. Without a form that contains the fields, those fields are invisible. In this respect, a Notes form is comparable to a Claris FileMaker layout.) However, an API application can read fields regardless of what form was used to create the document.

C API applications can generally be as efficient as the client and server itself – although, since the HiTest C and Notes C++ APIs are built on top of the C API, HiTest C and Notes C++ applications have an extra layer between them and the Notes Core, and do therefore incur some performance penalty. Figure 5 shows the layering of functionality in the Notes architecture.

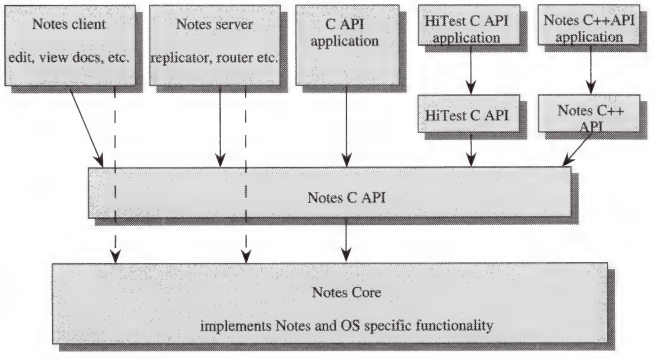


Figure 5. Notes architecture and the APIs

Notes C API

The API is implemented as a library of over 1000 C language functions, data structures, and symbolic constants to access Notes databases, network services, and administration services. The number of exposed functions from Notes Release 3 to Release 4 has increased by more than 50%, while maintaining compatibility with source code written for Release 3 API programs. The API is supported on every platform that supports a Notes client or Notes server.

Why use the Notes C API?

Given that both LotusScript and Agents can be used to automate the Notes client, and that a GUI interface can be built on top of a database, why should it be necessary to write a standalone Notes application in C? Notes API applications give the programmer flexibility and options that are not available in any other way.

There are many situations where it is appropriate to use the API, rather than developing internally in Notes.

- You may wish to create a customized user interface for a target environment.
- API programs do not require the Notes client application to be running. In contrast, to run applications using LotusScript or Notes database tools requires that the Notes client be running, which increases RAM overhead.
- The task may not be achievable using only the tools available within the Notes client.
- The application may need to integrate tightly with an in-house development environment other than Notes, or may need to integrate with off-the-shelf code libraries.

API capabilities

Among other things, the Notes C API can:

- create, delete, and copy Notes databases
- read, write, and modify Notes documents
- search, sort, and index documents in Notes databases
- send mail messages and documents
- read, write, and modify the design of a Notes database
- read, write, and evaluate formulas
- perform system Administration (invoke replication, modify the ACL, log user access)

API limitations

The Notes C API cannot:

- interact with the Notes user interface (e.g., it can't force Notes to open a window)
- access the Notes Desktop database
- run an API program if Notes is not installed

The C API toolkit

The API Toolkit includes the API header files and libraries, a large number of sample programs, and documentation. The sample programs show how to use the API functions, using a representative selection of the functions. The Toolkit documentation comes as two Notes databases, a *User's Guide* which introduces the concepts and functionality of the C API, and gives details on application development for each of the various supported platforms, and a *Reference Guide*, which explains every function, data structure, and symbolic constant used in the C API.

C API application development

C API applications for Mac OS are developed natively on a Macintosh. The official development environment is MPW using the Symantec C compilers for 68K and PowerPC. All development tools are included on the *Essential Tools and*

Objects (ETO) CD-ROM, published by APDA. Makefiles are included for all samples, as are MPW build scripts that automate the use of the makefiles. In addition to the Symantec compilers for MPW, the Metrowerks PowerPC compiler for MPW can be used, as can the Symantec Project Manager for PowerPC and the Metrowerks IDE for PowerPC. The Mac OS development environments are discussed in detail below.

Data Types

When talking about Notes standalone programmability, it may be useful to forget that the client application even exists, and think of Notes as an API to a database engine. In this paradigm, everything is a database. Even email can be accessed via database calls.

All API data structures are logical structures, and not necessarily physical structures. Programmers should not assume that any data structures in the API correspond to actual file formats. Notes database file formats are not publicly available.

Database header

Every database has a header which contains the database title, categories, access control list, replication history, and the user activity log.

Note (Document)

A note, more commonly called a document, contains its own header followed by any number of items, which are the actual data fields or items. (The terms *item* and *field* are sometimes used interchangeably.)

Notes canonical format

All Notes internal data is stored in Intel byte ordering with no pad bytes. Generally, Notes insulates API programs from the details of conversion. For certain data types, the programmer is responsible for the conversion. The Notes C API User's Guide documents this, and the sample programs in the C API Toolkit demonstrate how and when to do the conversion. In the future, the Notes C++ API will do all the conversion automatically, and byte ordering and alignment will cease to be an issue.

Each Notes server can support one or more databases. Each database can contain one or more documents. Each document can contain one or more items. Each item has a distinct data type. Data types fall into two categories: *simple* and *composite.*

Simple data types

Simple data types include:

- **Text and Text List**: e.g., subject field of a mail memo
- **Number and Number List**: e.g., part number or order number
- **Time/Date and Time/Date Range or List**: e.g., date field

Composite data types

Composite data types include:

- **Rich Text**. The functionality provided by Lotus Notes Rich Text is actually a superset of the industry standard (Microsoft RTF). However, the data structures used in Lotus Rich Text are different than those used in Microsoft Rich Text. The C API Toolkit contains sample programs and source code to create and manipulate Rich Text, and to call the import and export libraries to convert between Microsoft Rich Text and Lotus Notes Rich Text. Lotus Notes Rich Text may contain bitmaps and hyperlinks (also called doclinks) to other documents.
- **Object**: e.g., embedded OLE object, file attachment
- **Tables**: e.g., tabulated data, spreadsheet-like grid
- **User Data**: Programmers can define their own data types which are not interpreted by Notes, but are simply stored as a stream of binary data.

Searching a Database

Using the C API, there are four ways to search a Notes database.

- **Linear**: Scans all documents in a database, applying search criteria. This type of search should be used when the search criteria are not known until runtime. The linear search is accomplished using a C API function that takes as an argument a pointer to a function that the programmer defines, called an "action routine". The action routine is called for each document that matches the search criteria. Linear is an inefficient search method.
- **Indexed**: Uses views to sort and categorize documents. This type of search should be used when the order of matched items is important. At runtime the programmer may create the view to specify the hierarchy.
- **Full Text**: To be used if search keywords might be in Rich Text fields.
- **Direct Access**: Not a search, but if the Note ID of the document or the name of an item within a document is known, it can be accessed directly. In other words, if the programmer knows where the data is, the programmer is not forced to use the search API in order to access it.

Notes Security

Users want to know that their workstations are secure, and IS departments want to know that their servers and networks are secure. With all the talk about the Internet and its being perceived as having low security, developers can take advantage of the fact that Notes is secure.

Notes provides user identification, access control lists, and password protection as well as encryption. It is important to know how this affects API applications, in order to take advantage of the features of Notes security.

User ID (Notes ID)

Every user is assigned a "User ID", sometimes called a "Notes ID" (not to be confused with a "Note ID"; a User/Notes ID identifies a person, while a Note ID identifies a document or note). A User ID contains the user's name, a Notes license number, a certificate which enables access to servers which are set up to recognize that certifier, and both a public and private encryption key. To access databases, a user would need a User ID file plus a password. This is far more secure than merely typing a username and a password, since a User ID can only be transmitted electronically (or physically). Notes Release 4 adds an additional level of security over Release 3 by offering encryption on the workstation. An API program can use only one User ID at a time.

Access control lists (ACL)

An Access Control List is a list of users, specifying what level of access each user has to a database. Each entry in the ACL contains a user name, access level, flags, and roles. Every database must contain an ACL.

Server-level security

The user of an API program must have access to the server. An API program cannot open a database that the user could not have opened through the Notes user interface.

Database-level security

Database-level security is controlled by the ACL, which defines who can use a database and what operations they are permitted there.

View-level security

If the view has a read access list, then it is enforced in the API. If the user does not have access to the view, the error message will indicate that the view does not exist.

Form-level security

If a form has a read access list, then it is enforced in the API if the API program implements it. The API ignores the compose access list for forms.

Document-level security

The document read, editor, and author access control lists are enforced by the API. For example, if the user does not have read access to a document, `NSFNoteOpen()` will fail, and `NSFSearch()` will not find the document. If a user has read access but not author access, then `NSFNoteOpen()` will succeed, but `NSFNoteUpdate()` will fail and will return the error `ERR_NOT_AUTHOR`.

Section-level security

A form may contain one or more sections. A section is a field that contains a list specifying who can edit the fields that follow the section field. Sections are not a real security feature, though, and an API program can write to or append to any field in a document regardless of section information.

Field-level security

Fields may be encrypted, and it is the responsibility of the API application to decrypt the field.

Encryption

Every Notes user is automatically assigned public and private encryption keys. The public key is usually stored on the Notes server. The private key is usually stored on the user's hard disk. A "personal" key is just another name for the private key; the personal key is used when a user wants to encrypt a document that will be kept locally.

The public and private keys are used for sending encrypted email. Notes hides the complexity of this from the user, who merely clicks a checkbox to enable encryption. Email is encrypted using the sender's private key and the recipient's public key. When the recipient receives the email, it is decrypted using the sender's public key and the recipient's private key.

Notes supports attaching a signature to an email, which also makes use of public and private keys.

Domestic vs. international

Notes uses RSA encryption, which has been classified by the US Government as a military technology, and thus the North American version of Notes cannot be exported in its most secure form. To send an encrypted email or document outside of the US or Canada, the International English version of Notes must be used, which has a slightly less but still quite secure level of encryption.

Field-level encryption

Documents are not encrypted; fields are. It is best to think of the document as a collection of items, all of which have been set up as encryption items. For example, the "body" field of an email is encrypted, but the "to" and "cc" fields are not.

Notes C++ API

The C++ API is not yet released, but will soon be available as a general beta release. It is built on top of the C API, and provides true OOP, with all the benefits of C++. One implementation goal of Notes C++ was to minimize the performance overhead. Notes C++ is single-chain, not multiple-inheritance.

Developer benefits

These include:

- Reduced application code size. Much of the code that the C API leaves to the individual developer is handled by the C++ classes in its high-level functions and data abstractions.
- Reduced application debugging and maintenance time, as well as more readable code. Since it is C++, there is type-checking done at compile time. The C++ classes implement run-time exception handling that is supported via "catch" and "throw". In addition, "failable" functions return status codes, which may be used both as information and to indicate an error.
- Sophisticated memory management. If an object allocates memory, it is responsible for freeing it. If the programmer allocates memory, the programmer is responsible for freeing it. If the programmer allocates an object but forgets to free it, it is automatically freed when the object goes out of scope.
- Ease of learning. The C++ classes are easier to learn than the hundreds of C API functions or the HiTest C objects.
- Rapid development and deployment.
- Multi-platform support. The API will support every platform that the Notes C API supports.

Searching a database

The C++ API has all of the searching capabilities of the C API, plus more. The C++ API–specific navigation includes:

```
// Iterators (class maintains the state)
ACLEntry = ACLEntryIterator.GetNextEntry();

// Arrays (indexed access to data)
Doc = Documents[i];

// Named data can be directly accessed
Status = Database.GetForm("Discussion Topic", &Form);
```

Vendor-Independent Messaging

The Vendor-Independent Messaging (VIM) Interface began as an industry-standard, platform-independent application programming interface specification for messaging systems. It has evolved into the Lotus email standard interface for Notes and ccMail. In addition, various Lotus products have different levels of support

for such interfaces as MAPI (Messaging Application Programming Interface), CMC (Common Mail Call), and SMI (Simple Messaging Interface). In future releases, Notes will support POP (Post Office Protocol) and SMTP (Simple Mail Transfer Protocol), but it would be very premature to discuss the level of support for these protocols that will be available through the API.

The VIM API enables programmers to write applications that link the functionality of messaging systems to the functions provided by end-user applications. With VIM, the resulting mail-enabled or mail-aware applications can be developed over a wide range of platforms. VIM provides a single interface that allows programmers to combine the services of various messaging systems into one application. Using the VIM API functions, one can design a number of specialized mail-enabled applications. For example, a programmer could develop an application designed to analyze real-time stock prices and send an email when a particular stock reaches or drops to a certain price.

The Lotus VIM Developer's Toolkit provides header files, libraries, documentation, and sample programs, and is compatible with Notes-based electronic mail.

VIM does a few things that cannot be done using the C API, and does many things in a manner that is more convenient for the programmer. The C++ API will have a class implementing basic email capabilities; however, keep in mind that the C++ API is built on top of the C API, not VIM.

Some of the most important features of VIM include its ability to:

- compose and send messages with file attachments
- receive, store, and process delivered messages
- search and modify address books
- interoperate with ccMail

Leveraging Apple Technologies

Newton

The Notes C API Toolkit currently contains one Newton sample called MakeBook, and will soon have more. The sample program exports the Notes C API User's Guide database into a form that can be transferred to the Newton and be read by the built-in Newton Book browser. This requires the use of an Apple-supplied Newton Book "compiler" called Book Maker, plus the Newton Developer Toolkit to build the output of Book Maker into a package and to download that package to the Newton. This program was demonstrated at the LotusSphere '96 conference in Orlando. Browsing the Notes C API User's Guide on the Newton is actually faster than browsing it using Notes Release 4 on a PowerPC or a Pentium-based machine, even if the database is stored locally on that machine. (Let us, however, keep this in context. MakeBook, though it could be modified to export any Notes database, is not a general-purpose Notes database export tool for end users, but rather a technology demonstration.)

DILs

Apple has recently released the Newton Desktop Integration Libraries (DILs) to enable developers to write Mac OS and Windows applications that exchange data with the Newton. In the future, the Notes API group may be releasing sample programs that show how to leverage the DILs in Newton/Notes applications. Recall that the MakeBook sample uses another program to transfer the data to the Newton from the Mac or Windows machine; functions in the DILs might handle that task.

OpenDoc

OpenDoc has the potential to be a key Notes development tool, since OpenDoc's concept of containers and parts is somewhat analogous to the Notes paradigm of the Notes client application and runtime services implemented by databases, LotusScript, and API programs. The Notes API group is looking at ways to integrate the Notes API into OpenDoc to provide sample OpenDoc parts in the Lotus Notes C API Toolkit. Nothing has been planned at this time, but keep in mind that the Windows and OS/2 versions of OpenDoc are being developed by IBM, which recently acquired Lotus.

AppleScript

AppleScript can be used to make an application programmable by end users, and can be used to implement programmability between applications. Although the Notes client application is *not* AppleScript-aware, it is possible to create an AppleScript-aware application that provides an interface between AppleScript and the Notes C API. (This is

very different from scripting the Notes client, a task that should be done using LotusScript.) For example, imagine implementing the AppleScript database suite, or some variation of it, which uses Notes as its database, as implemented via the C API. Another idea would be to use AppleScript to implement a layer on top of VIM, to make it easier for developers to mail-enable their applications. Lotus is also working on a "glue layer" interface between LotusScript and AppleScript.

Mac OS Development Environment

Shared library technology

The C API is implemented as a shared library on Mac OS (as on UNIX; it is implemented as a number of DLLs on Windows and OS/2). Although the library is linked with the application, the actual code does not get copied into the application. If more than one application uses the same shared library, only one copy needs to reside on the user's machine. Both Notes and the API application call into the same copy of the shared library code. In addition to saving disk space, using shared libraries for Lotus Notes ensures that both Notes and API applications will always be running with the same version of the libraries.

For a long time, Apple did not provide a shared library technology for developers to use. This is the main reason why Notes Release 3 did not have a developer-accessible API. Then Apple implemented the Apple Shared Library Manager (ASLM) for 68K Macintosh models *only*, followed by the Code Fragment Manager (CFM) for PowerPC Macintosh models *only*. Thus, when the Notes Release 4 project began, there was no choice but to implement the 68K version with ASLM and the PowerPC version with CFM.

Today, both ASLM and CFM are available on both platforms, but too late for Notes Release 4. While this has no impact on Notes users or on Notes API code, it does have an impact on what linkers can be used to develop a Notes API application. Restriction of linker choices restricts compiler choices, and development environment choices.

MPW support

The current release of the API Toolkit for Mac OS recommends the use of MPW with the Symantec compilers for 68K and PowerPC. It is possible to use the Metrowerks compiler for PowerPC, but not the Metrowerks compiler for 68K, since Metrowerks does not support linking ASLM shared libraries. It is not possible to use Metrowerks to build 68K Notes API applications.

The C API Toolkit includes an MPW UserStartup file and a number of script files and makefiles, to ease development of the sample applications, and to provide a foundation for developing new projects. The C API User's Guide that is included with the Toolkit provides extensive documentation on setting up the development environments for each platform that the Notes API supports.

Symantec and Metrowerks project environment support

Notes C API PowerPC applications can be built using the Symantec Project Manager. Notes C API PowerPC and 68K applications can be built using the Metrowerks IDE. Unfortunately, using Symantec Think C for building 68K Notes API applications is extremely difficult, and is not something that is being looked into further at this time.

Information on using the Symantec Project Manager and Metrowerks IDE for Notes C API development can be found at http://www.Lotus.com/devtools/21da.htm, as well as in the documentation that is included with the C API Toolkit.

Support of the Symantec and Metrowerks Project environments for the Notes C++ API Toolkit is a goal, since this will enable developers to use the class browsers, as well as easing the integration with the Symantec and Metrowerks class libraries.

Code Sample

Disclaimer: All of the header files, some macro definitions, and lots of error checking have been removed in an effort to make this code simpler, shorter, and easier to read. The full (and working) source code for this program is included on the Notes C API Toolkit CD-ROM, along with makefiles and build instructions for Mac OS 68K and PowerPC, and Windows 16- and 32-bit.

Listing 1: MakeBook.c

Globals

```
FILE   *outfile = NULL;
long   uniqueLabelCounter = 0;
char   *field_text;
```

Main

This standalone program is declared with argc and argv, even though these arguments cannot be used in the Mac OS, unless it were to be built as an MPW tool. Aside from the conditional code for initializing the Mac OS toolbox, all of this source is 100% platform independent. (The actual sample code on the Notes C API Toolkit CD ROM contains a macro that hides platform specific initialization.)

All output from this program goes to a text file, which is created and written using the standard C file i/o routines. Again, this choice of file i/o was made for platform independence. In Notes API applications, one may choose to use conditionally compiled code to add platform specific file system functionality such as Mac OS file types.

`NotesInitExtended()` is a Notes API function that initializes the Notes runtime library. At the end of `main()`, `NotesTerm()` is called. Every standalone program must have one set of init and term calls. Multi-threaded applications must have one set of calls for each thread. In this program, `NotesTerm()` is called as part of the macro `API_RETURN()`.

The database is opened using `NSFDbOpen()`, processed with the function `ReadDatabase()`, and then closed using `NSFDbClose()`. The open function returns a "handle" to the database, which is then used by all functions that refer to that database. `NSFDbOpen()` takes the name of database as a full pathname, or as a filename if that database is stored locally. If the database is stored on a Notes server, there is an API function that will build the pathname.

```
void main(int argc, char *argv[])
{
  DBHANDLE db_handle;         // handle of source database
  STATUS   error = NOERROR;   // return status from API calls

#ifdef MAC
    InitMacToolBox();
#endif

  if ((outfile = fopen("book.src", "w")) != NULL)
  {
    //
    // Output data that is required by the Apple Book Maker application, for the
    // creation of a Newton Book.
    //
    // For more information, please refer to the "Newton Book Maker User's Guide",
    // which is included with the Apple's Newton Toolkit for Mac OS and Windows.
    //
    fprintf(outfile,
"\
.isbn Notes:LOTUS\n\
.date 02/05/96\n\
.author Rick Gansler\n\
.publisher Lotus Development Corp.\n\
.copyright (c) 1996 Lotus Development Corp.\n\
.shorttitle API User\n\
.title Lotus Notes - API User's Guide\n\
.blurb\n\
The Lotus Notes API User's Guide exported from Notes to
Newton Book format.\n\
.layout Indented 1 Sidebar 11\n\
.layout TitlePage 12 NoTitle\n\
.layout ContentsPage 2 Sidebar 10 Main NoTitle\n\
.layout Default 12\n\
");

    // Initialize the Notes runtime library
    if (NotesInitExtended(argc, argv))
      NOTES_INIT_ERROR;

    // Open the database
    if (error = NSFDbOpen("API40UG.NSF", &db_handle))
      API_RETURN(ERR(error));

    field_text = (void *)malloc(32000);

    // Generate the Newton Book contents
    uniqueLabelCounter = 0;
    error = ReadDatabase(db_handle);
    if (error)
    {
      NSFDbClose(db_handle));
      API_RETURN(ERR(error));
    }

    // Close the database
    if (error = NSFDbClose(db_handle))
      API_RETURN(ERR(error));

    fclose(outfile);
    free(field_text);
  }

  API_RETURN(NOERROR);
}
```


ReadDatabase

`ReadDatabase()` takes a database handle, and reads each document in the database. In this case, the database has a view that is defined in the database. One could also create a custom view at runtime. The pairing of a view and a collection is analogous to a search result that has been sorted. Once the collection has been opened, `ReadEntries()` is called to return a buffer containing a reference to each document in the collection, plus a counter that indicates how many documents are in the collection. Each document is identified by its unique note id. A loop is used to iterate through the buffer and pull out each note id, which is passed to the function `ReadNote()`.

```
STATUS ReadDatabase(DBHANDLE db_handle)
{
  STATUS        error=NOERROR;  // return status from API calls
  NOTEID        ViewID;         // note id of the view
  HCOLLECTIONh  Collection;     // collection handle
  COLLECTIONPOSITION CollPosition;  // index into collection
  HANDLE        hBuffer;        // handle to buffer of info
  DWORD         EntriesFound;   // number of entries found
```

```
WORD        SignalFlag;     // signal and share warning flags
BYTE        *pBuffer;       // pointer into info buffer
DWORD       I;              // a counter
NOTEID      EntryID;        // a collection entry id

// Get the note id of the view we want
if (error =
      NIFFindView(db_handle,"TABLE OF CONTENTS",&ViewID))
  return(error);

// Get the current collection using this view
if (error = NIFOpenCollection(
        db_handle,      // handle of db with view
        db_handle,      // handle of db with data
        ViewID,         // note id of the view
        0,              // collection open flags
        NULLHANDLE,     // handle to unread ID list (input & return)
        &hCollection,   // collection handle (return)
        NULLHANDLE,     // handle to open view note (return)
        NULL,           // universal note id of view (return)
        NULLHANDLE,     // handle to collapsed list (return)
        NULLHANDLE))    // handle to selected list (return)
    return(error);

  // Set a COLLECTIONPOSITION to the beginning of the collection
  CollPosition.Level = 0;
  CollPosition.Tumbler[0] = 0;

  // Get the note ID and summary of every entry in the collection. In the
  // returned buffer, first comes all of the info about the first entry, then
  // all of the info about the 2nd entry, etc. For each entry, the info is
  // arranged in the order of the bits in the READ_MASKs.
  do
  {
    if (error = NIFReadEntries(
        hCollection,        // handle to this collection
        &CollPosition,      // where to start in collection
        NAVIGATE_NEXT,      // order to use when skipping
        1L,                 // number to skip
        NAVIGATE_NEXT,      // order to use when reading
        0xFFFFFFFF,         // max number to read
        READ_MASK_NOTEID,   // info we want
        &hBuffer,           // handle to info buffer (return)
        NULL,               // length of info buffer (return)
        NULL,               // entries skipped (return)
        &EntriesFound,      // entries read (return)
        &SignalFlag))       // share warning & more signal flag return
    {
      NIFCloseCollection(hCollection);
      return(error);
    }

    // Check to make sure there was a buffer of information returned
    if (hBuffer == NULLHANDLE)
    {
      NIFCloseCollection(hCollection);
      NSFDbClose(db_handle);
      return(NOERROR);
    }

    //
    // Lock down (freeze the location) of the information buffer. Cast
    // the resulting pointer to the type we need.
    //
    // OSLockObject()  is sort of like the Notes equivalent of locking a
    // Mac handle and then dereferencing it. However, Notes abstracts
    // memory management, since each Notes platform may implement it
    // differently. For example, a Windows handle is a very different kind
    // of data object than a Mac handle.
    //
    pBuffer = (BYTE *) OSLockObject (hBuffer);

    // Start a loop that extracts the info about each collection entry from
    // the information buffer
    for (i = 1; i <= EntriesFound; i++)
    {
      // Get the NoteID of this entry
      EntryID = *(NOTEID*)pBuffer;

      // Advance the pointer over the NoteID
      pBuffer += sizeof(NOTEID);

      if (! (NOTEID_CATEGORY & EntryID))
        ReadNote(EntryID, db_handle);
    }

    // Unlock the list of NoteIDs.
    OSUnlockObject(hBuffer);

    // Free the memory allocated by NIFReadEntries
    OSMemFree(hBuffer);
  } while (SignalFlag & SIGNAL_MORE_TO_DO);

  // Close the collection
  error = NIFCloseCollection(hCollection);

  return(error);
}
```

ReadNote

`ReadNote()` takes a note id and a database handle, opens the document referred to by the note id, reads only the fields in that document that are needed, formats them, and outputs them to a text file in a form that can be imported by the Apple program Book Maker.

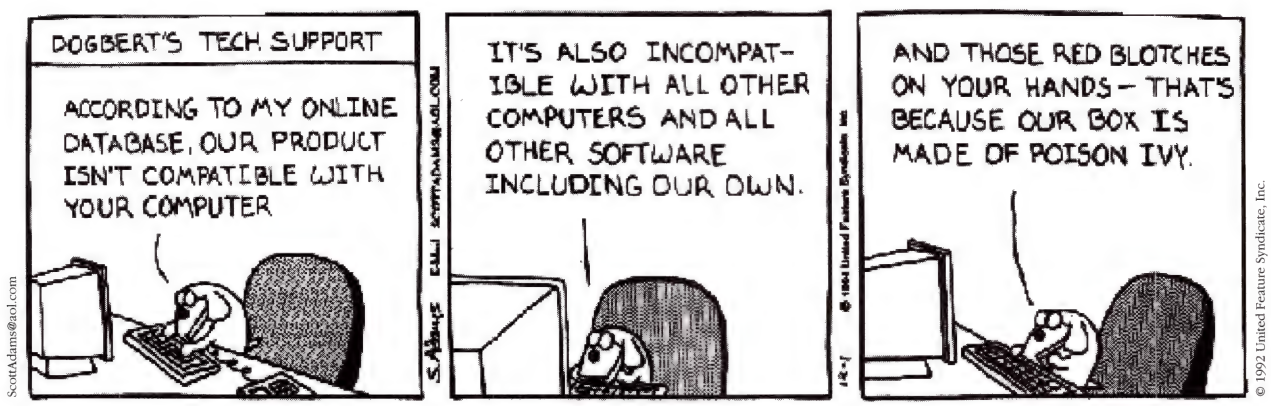
```
STATUS far PASCAL ReadNote(NOTEID noteID, DBHANDLE db_handle)
{
  NOTEHANDLE    note_hdl;
  WORD          field_len;
  long          sectionNum_num;
  long          chapterNum_num;
```

```
char          sectionName_text[100];
char          titleName_text[100];
STATUS        error;
static long   prevSectionNumber = -1;  // this is a static

// Open the document whose note id was passed to this function
if (error = NSFNoteOpen(db_handle,  noteID, 0, &note_hdl))
  return(ERR(error));

sectionNum_num = NSFItemGetLong(note_hdl, "SectionNumber", 0L);

chapterNum_num = NSFItemGetLong(note_hdl, "Chapter", 0L);

field_len = NSFItemGetText(note_hdl, "SectionName",
                           sectionName_text,
                           sizeof (sectionName_text));

field_len =
    NSFItemGetText(note_hdl, "title", titleName_text,
    sizeof(titleName_text));

// If this document is the first in a new chapter, create a new heading in the
// Newton Book Overview (or table of contents)
if (sectionNum_num > prevSectionNumber)
{
  prevSectionNumber = sectionNum_num;
  fprintf(outfile,
       ".subject 1 startspage centered name=%ld\n",
       (long)uniqueLabelCounter++);
  fprintf(outfile,
       "(%ld) %s\n",sectionNum_num,sectionName_text);
}

// Every document gets an entry in the Newton Book Overview.
fprintf(outfile,
    ".subject 2 name=%ld\n", uniqueLabelCounter++);
fprintf(outfile,
    "(%ld) %s\n", chapterNum_num, titleName_text);

// Body of the document (Notes Rich Text field)
if (NSFItemConvertToText(note_hdl, "text", field_text,
                         FILE_SIZE, '\0') > 0)
{
  // If the data from Notes contains a \r\n, then remove the \r.
  // The code for this function is not included in this listing.
  ProcessLineFeeds(field_text);

  // If the data from Notes had a line that began with a period, that would
  // confuse Book Maker, so we replace the period with a space.
  // All Book Maker commands begin with a period at the start of a line.
  // The code for this function is not included in this listing.
  CorrectLinesStartingWithPeriod(field_text);

  fprintf(outfile, "%s\n", field_text);
}

// Close the note
if (error = NSFNoteClose(note_hdl))
  return(ERR(error));

return (NOERROR);
}
```

SUMMARY

Lotus Notes Release 4 presents the Macintosh community with a very exciting opportunity. First, Release 4 is a complete overhaul of the user interface and functionality of Release 3, so should be well received by Mac users. Second, Release 4 exists in both 68K and PowerPC-native versions. Lastly, current support for LotusScript and the C API, and future support for VIM and the C++ API, give Macintosh developers the tools to bring custom Notes-based solutions to the Macintosh platform, as well as to take advantage of Apple technologies by incorporating them into custom Notes applications.

Dilbert® *by Scott Adams*



Reprinted by permission of UFS, Inc.

Continued from page 27

Dave: ANSI C and ANSI C++ are two different languages, yet there is a single front-end to handle both. How does this work?

Andreas: ANSI C++ was derived from ANSI C, and most of its new features are really add-ons. Almost all ANSI C features are also supported in C++, so I was able to support both languages from the same front-end by disabling C++ features depending on the state of a global variable. There are some syntactic and semantic differences, but I was able to code around those with some "if-else" statements. This really makes adding features or fixing bugs a lot easier, because everything will only have to be done once. What's also really interesting is that our compiler architecture allows us to treat both C and C++ with the same front-end, making the transition from C to C++ much easier because it's the same compiler. Just flip a switch and write your code.

Dave: When you first started on your compiler, work on a C++ standard had just begun (with the publication of the *Annotated C++ Reference Manual*, or *ARM*). How has this process evolved?

Andreas: The *ARM* was the only real useful C++ language reference when I started. This book was used as the base document for the ANSI C++ standard. It has a lot of gray areas, and the template chapter is really vague, but it has some useful sections that explain how certain C++ features like virtual functions can be implemented. Unfortunately, those sections have been removed from the standard.

The current ANSI C++ draft is the size of a phone book. Many features (namespaces, RTTI, bool/true/false, a complete C++ library) have been added to the original definition, and there are three or four revisions every year. It is very hard to keep track of all these changes, and I think it will take another two years until there will be a final ANSI C++ standard.

UNIFORM RESOURCE LOCATORS

By Jim Straus, URLs@mactech.com

Don't hesitate to notify me of any sites that you think would be of interest! As always, the full list is maintained on-line at http://www.mactech.com/URLs.html.

Web Watch

This being September, traditionally the time for going back to school, we will look at some Internet sites which allow you to learn to program your Macintosh on your own – and which also take advantage of the Internet as a publishing medium.

An excellent place to start if you want to program the Mac is the Macintosh C Home Page. This is a book, by K. J. Bricknell, in Australia, covering everything from low-level events through custom windows. Included are sample code, demonstration programs and lots of other stuff. There are two versions, one for CodeWarrior and one for Symantec; both can be downloaded, and the former can be read on-line, with many nice Web touches such as two frames with code and commentary respectively.

Macintosh C
http://www.AmbrosiaSW.com/alt.sources.mac/macintosh-c/

The Mac OS Students and Teachers site provides links to self-study tutorials and other on-line resources (including commentary on the Macintosh C Home Page!), plus Internet classes, where a group of people will follow one of the self-study courses with a mentor providing guidance. If you need more structure to your learning, this would be an excellent way to proceed. And if you can give of your time to help those still learning, be sure to stop in. We need to evangelize Mac development and this would be a great way to help.

Mac OS Students and Teachers
http://aimed.org/aimed/most/

No discussion of on-line learning would be complete without stopping by Apple's Developer University. Developer University has created self-study courses that have been available in printed form for years. They have started to move these courses to the Internet. Four of the courses have been transformed, and more are promised.

Apple Developer University
http://dev.info.apple.com/du/7.5online/7.5TopicsText/ST01-7.5Intro.html

To learn about Symantec's THINK Class Library, there is a very nice tutorial, again out of Australia. The student is taken through the development of several programs, creating the projects, using Visual Architect, and writing the code.

Symantec TCL Tutorial
http://www.cs.uow.edu.au/people/nabg/SymTute/SymTCL.html

For PowerPlant learners, there is the PowerPlant Beginners page. This isn't so much a tutorial as a collection of reference material about PowerPlant. When the Beginners page originally appeared, it looked like it was going to be collaborative effort between the author and his readers. It has turned into more of a place for the author to report what he has discovered about PowerPlant, which is currently rather brief and superficial.

PowerPlant Beginners
http://www.interlog.com/~breakpt/html/powerplant.html

And let's not forget good old *Inside Macintosh*. Apple has made some of the volumes available on the Internet, as a publishing test. Hopefully they will complete this project, as it would be very useful to be able to use the Internet to look up that tidbit you need to finish your project. In any case, you will end up reading a lot of *Inside Macintosh* if you do much Mac programming at all.

Inside Mac
http://dev.info.apple.com/insidemac/files/Contents.html

Thanks this month to Jonn Chard, Evan Garbe, Elliotte Rusty Harold, Devon Hubbard, Carol Lashman, Steve Makohin, Cliff Miller, Dave Nebinger, D. Slattery, Tom Nielsen, Wayne Walrath, and many others for their contributions for their suggestions and pointers to new and old sites.

Quickies

Internet-Related

Mac Internet Software	http://www.cyberatl.net/%7Emphillip/index.html

New Technologies

Clock Chipping	http://violet.berkeley.edu/~schrier/mhz.html
Cryptography Interface	http://www.io.com:80/~combs/htmls/mcip.html

Vendors, Products and Miscellaneous

Cyberdog Demonstration Page	http://www.resnova.com/cyberdog/
Info-Mac Home Page	http://www.pht.com/info-mac/
Internet Only Mac Users	http://www.uta.edu/acs/microsys/mac/.HOME/jstewart/IO-MUG.html
The Ultimate Newton	http://rainbow.rmii.com:80/~rbruce/

MacRegistry™

Developer Job Opportunities

If you are a Macintosh developer, you should register with us! We have a database that enables us to let you know about job opportunities. When we are asked to do a search by a client company the database is the first place we go. There is no charge for registering. The database service is free. Geographic Coverage is nationwide.

Marketability Assessment - To get a specific feel for your marketability send a resumé via Email or call. You may also request a Resume Workbook & Career Planner.

Discreet - We are very careful to protect the confidentiality of a currently employed developer.

Scientific Placement is managed by graduate engineers, we enjoy a reputation for competent & professional job placement services and we are Mac fanatics.

1-800-231-5920 • das@spi.com • Fax 1-800-757-9003
http://www.scientific.com

Scientific Placement, Inc.

MT, Box 19949, Houston, TX 77224, 713-496-6100 Fax: 713-496-0373
MT, Box 71, San Ramon, CA 94583 510-733-6168 beth@spica.bdt.com
MT, Box 202676, Austin, TX 78720-2676 512-260-0123 lej@zilker.net
AppleLink: D1580; Compuserve: 71250,3001; AOL: davesmall

THE TRATTNER NETWORK

COMPUTER CONSULTING SERVICES

The Trattner Network (TTN) has the best opportunities for Macintosh developers in Northern California and Nationwide.

TTN represents clients with projects in Internet, Multimedia, Codewarrior and OpenDoc development among others.

Now you can visit us our web site at www.tratnet.com and see what openings are available for....

- **Software Developers**
- **Hardware/Firmware Engineers**
- **QA/QC Professionals**
- **Project Coordinator/Managers**
- **Multimedia Developers**
- **Network Professionals**

The Trattner Network has a unique history in **Mac consulting** coupled with exposure to emerging technologies. If you are looking for a chance to enhance your skills and marketability, **please send, fax, e-mail or link your resume to:**

The Trattner Network
Attn: Emily Hoolhorst
170 State Street, Suite 240 • Los Altos, CA 94022
Phone: 415•949•9555 ext.115 ; Fax: 415•949•1026
AppleLink: trat.net ; E-mail: emily@tratnet.com
http://www.tratnet.com

THE DIGITAL TALENT SOURCE

PARALLEL SOFTWARE

CHICAGO AREA

Parallel Software is looking for software engineers who are serious about component based **OPENDOC** development. We are interested in Macintosh or Windows people with the following experience:

"C" or "C++", Pascal, OPENDOC
Newton, NDK, or JAVA
MacApp,Code Warrior, MacToolbox
Oracle, Sybase, or SQL Databases

Please write or phone:

Mr. John D. McMahon
Parallel Software, Inc.
608 S. Washington, Ste. 101
Naperville, Illinois, 60540
Tel (708)369-0100
Fax: (708)355-7870
psgmcmahon@aol.com.

MacTech Magazine is your recruitment vehicle

When you need to fill important positions at your company, MacTech Magazine is the consistent choice of companies across the country for hiring the best qualified Macintosh programmers and developers. Let MacTech Magazine deliver your recruitment message to an audience of over 27,000 qualified computer professionals.

Call Ruth Subrin at
805/494-9797

Do you have a service that you want to advertise to over 30,000 Macintosh programmers and developers?

Do it in the
MacTech Magazine Classifieds –
an inexpensive vehicle
that *gets results.*
Call Ruth Subrin
at 805/494-9797
to reserve space NOW!

By Jon Kalb

DebugStr, the Modern Way

Capturing your program's iostream of consciousness

I have a friend who spends most of his time in cutting-edge (read, "reliable tools not yet available") environments. He likes to say: "There is no debugger; there is only DebugStr." While he may stretch the truth, he has a point. Our friends who build source-level debugging tools are a heroic lot, but there are times when MacsBug is the only option. This article, along with the accompanying `dout` Library, is one attempt to make using `DebugStr` a little more modern.

Using `DebugStr` is usually a clumsy process of converting integer values to strings and concatenating Pascal strings. There had to be a better way, I thought. That's when it occurred to me to implement the low-level debugger as a C++ output stream.

The Solution Domain

This approach may be valuable for two groups: fellow users of `DebugStr`, and individuals interested in using and/or implementing C++ streams. But first, some limitations:

- I use the term "library" rather loosely. The `dout` Library is really just a small collection of source files. I also include project files for both Metrowerks and Symantec, to demonstrate how to use the library.

- In no way does this library replace a source-level debugger. If it is possible for you to use one, do so.

- Although, for the sake of clarity, I don't follow this practice in this article, in real code your should always surround every use of the `dout` stream with some type of `#ifndef NDEBUG` statement. (`NDEBUG` is the preprocessor variable that controls the `assert` macro found in assert.h. Define `NDEBUG` for shipping code, and leave it undefined for code under development.) To make this task a little easier, the `dout` Library includes the data statement macro, `ds`. For example, the line "`ds(dout << myObject);`" would be completely stripped if `NDEBUG` was defined, and would become "`dout << myObject;`" if `NDEBUG` wasn't defined.

- Streaming the semicolon character `';'` to `dout` is problematic. MacsBug interprets the text following a semicolon as a MacsBug command, and attempts to execute text that you intended to display.

- Below are examples of overloading the insertion operator for objects to support debugging in a streams environment. This is possible if your code base doesn't already use insertion operator overloading for other purposes. It may be possible to use the same routine for, say, storing an object to disk and for debugging purposes, but it seems unlikely. I suggest a work-around later.

- Streaming data to `dout` in the constructor of a `static` or global object (an object that is constructed before the first line of `main` is executed) is problematic. More on this below.

Jon Kalb works for Liberty Software, an exclusively Macintosh software contracting firm. Jon is currently on the Harry Browne for President campaign staff. You can write to him at kalb@libertysoft.com.

Using the Library

To use the dout Library you must include the standard C++ iostreams and the source file **debugbuf.cp**. Each source file that writes to the dout stream must include the header file **doutstream**.

Standard Streams

You can use dout just as you would cout. For example:

```
#include "doutstream"
//...
dout << "file corrupted at offset " << byteOffset << endl;
//...
dout << "name: " << un << endl << "password: " << pw << endl;
//...
dout << "completed pass " << i << " of " << total << endl;
//...
```

You can use dout just like any standard C++ output stream, but, since the debugger is more than just a byte sink, we should be able to do more, and we can.

The dout Difference

The library defines a set of symbols that cause the stream to behave in interesting ways. Normally, the stream buffers characters until it fills a line (arbitrarily defined as 60 characters) or until the caller streams an endl. You can flush this buffer at any time by streaming an endl. (endl is the standard C++ streams manipulator function for advancing to the next line.) dout produces a blank line if there is currently no text in the stream's buffer. To flush the buffer without creating blank lines, stream the constant doutsoftflush ('\r').

dout supports both horizontal and vertical tabs when streaming douttab ('\t') or doutverttab ('\v'). Streaming doutformfeed ('\f') flushes the buffer and inserts a line of underscores. Streaming doutbackspace ('\b') "eats" the last character in the buffer. Streaming doutsysbeep ('\a') calls SysBeep; however, just like other data, this is buffered. (To beep immediately, follow doutsysbeep with endl.)

Streaming data to dout doesn't normally suspend your program the way that a call to DebugStr would. If you want to drop into the debugger, then stream either doutdebug ('\0') or doutdropin ('\0'). These are synonyms; either one will flush the buffer and leave you in the debugger with your application suspended. (Streaming the End-of-Text character ('\x03'), usually associated with the Enter key, will do the same thing.)

As a side-effect of using MacsBug for streaming output, we can execute MacsBug commands. If a semicolon appears in the data stream, MacsBug attempts to execute the text that follows as a MacsBug command. (This can produce unexpected results if the data you are streaming just happens to contain a semicolon.) Several commands can be executed in sequence as long as each command is preceded by a semicolon.

Although you can use the dout Library in conjunction with source-level debuggers (products from Jasik, Metrowerks and Symantec support capturing messages sent with DebugStr), only MacsBug is going to be able to interpret and execute MacsBug commands.

This is an example of using dout to execute MacsBug commands.

```
dout << doutcommand << "hc" << doutsoftflush;
//...
dout << doutcommand << "dm #";
dout << (unsigned long)aPointer << endl;
```

Note the use of the constant doutcommand (';'), and that the stream buffer is flushed (with either endl or doutsoftflush) after each command. In this example, MacsBug executes the Heap Check command, and displays memory from the location pointed to by aPointer. The default behavior for commands is to execute the command and continue program execution. In the case of Heap Check, MacsBug suspends the execution of your program if the Heap Check finds that the heap is corrupted.

For my purposes this alone would justify using the library. The library handles conversion of integer values, there is no fooling with Pascal strings, and, the C++ implementation gives us type checking without worry. But, as they say on the infomercials: Wait, there's more.

Extending Streams

Using C++ means that we also get extensibility.

I have included **streamstructsmac.h** and **streamstructsmac.cp** as an example of how to stream standard Macintosh types (or any structure). This is not an attempt to define insertion routines for all Macintosh Toolbox structures, only an example to show you how to declare and define insertion routines for structures that you may find useful.

Listing 1: streamstructmac.h

Declarations of Insertion Operators for Point, Rect, and BitMap

The following declarations allow Points, Rects, and BitMaps to be streamed out.

```
#include <iostream.h>
#include <QuickDraw.h>

// inserters

ostream &operator<<(ostream &stream, const Point &rhs);

ostream &operator<<(ostream &stream, const Rect &rhs);

ostream &operator<<(ostream &stream, const BitMap &rhs);
```

This is the standard way to implement insertion operator overloading to standard C++ library streams. Note that there is nothing specific to the dout Library in this set of declarations (or even in the implementation that follows). These same routines could be used to stream structures of these types to any C++ ostream.

Listing 2: streamstructmac.cp

Definitions of Insertion Operators for Point, Rect, and BitMap

The following definitions allow Points, Rects, and BitMaps to be streamed out.

```
#include "streamstructsmac.h"

// inserters

ostream &operator<<(ostream &stream, const Point &rhs)
{
  stream << ".v(" << rhs.v << ") ";
  stream << ".h(" << rhs.h << ") ";
  return stream;
}

ostream &operator<<(ostream &stream, const Rect &rhs)
{
  stream << ".t(" << rhs.top   << ") ";
  stream << ".l(" << rhs.left    << ") ";
  stream << ".b(" << rhs.bottom<< ") ";
  stream << ".r(" << rhs.right << ") ";
  return stream;
}

ostream &operator<<(ostream &stream, const BitMap &rhs)
{
  stream << ".baseAddr(" << (void *)rhs.baseAddr << ") ";
  stream << ".rowBytes(" << rhs.rowBytes  << ")\n";
  stream << ".bounds(" << rhs.bounds    << ") ";
  return stream;
}
```

Note that the `BitMap` insertion function calls the `Rect` insertion function without any special syntax. C++'s ability to extend the language syntax to user-defined types allows this.

The style that I use shows the field names (preceded with a `'.'`) followed by the value of the field in parentheses. Sample output for a `Point`, a `Rect`, and a `BitMap`, might look like this:

```
.v(1) .h(2)
.t(3) .l(4) .b(5) .r(6)
.baseAddr(0x12345678) .rowBytes(7)
.bounds(.t(3) .l(4) .b(5) .r(6) )
```

This is all you need to know to use streams with `structs` or `classes` that either have no `private` or `protected` members or provide accessors for all such members. But what if you want to stream `private` members? This situation will arise often as you develop your own classes. The solution is to implement the insertion operator overload as always, and declare it as a `friend` function when declaring your class. A trivial but complete example of this is included in the source files as the class `privateMembers`.

Suppose that you have already defined insertion operator functions for some purpose other than debugging. For example, if you are streaming objects for persistent storage, the streamed form of your objects may not be what you would like to see streamed into the debugger. One way to work around this problem is to subclass the `ostream` class and make `dout` an object of this new subclass. Now, instead of defining the insertion operator in terms of an `ostream`, create an insertion operator function based on the new subclass.

The Implementation

The first thing to know about the implementation of `dout` is that `dout` is a standard C++ library `ostream`. Stream objects are responsible for taking inserted data, formatting it, and passing it to objects of type `streambuf`. It is the `streambuf` object that decides where the data ultimately ends up. By subclassing `streambuf`, we make `dout` possible. The library contains a `debugbuf` class that inherits from `streambuf`.

The following code, from the top of the **debugbuf.cp** file shows the relationships of these objects:

```
static debugbuf debugstream;
ostream dout(&debugstream);
```

`dout` is a standard `ostream`. Like all `ostreams` it requires a `streambuf` object to function; so, in the constructor, we pass the address of an object of type `debugbuf`, which is derived from `streambuf`. All of our work is done in the object of type `debugbuf`.

Class Declarations

The first thing to notice about the declaration of the `debugbuf` class is that no destructor or constructors are declared. Let's ignore this for the moment and look at the virtual functions inherited from `streambuf` in the `protected` section.

Listing 3: debugbuf.h

Declarations of debugbuf and debugbufinit

For debugbuf, virtual member functions of streambuf are declared as well as private members required for implementation. For debugbufinit the declaration does not have a memory footprint.

```
class debugbuf: public streambuf
{
  public:

    //debugbuf();
    //virtual ~debugbuf();

    // Construction and destruction are really handled by class debugbufinit.
    // Initialization is done in the private routine init().

  protected:

    // These protected member functions are virtual functions in the parent
    // (streambuf) class that we override to create our behavior.

    // The name overflow may be confusing - this just outputs a character to the
    // stream.
    // Calls outputchar().

    virtual int overflow(int c = EOF);

    // Since this is not an input stream, we want both pbackfail() and underflow() to
    // return EOF (thus indicating failure). It turns out that the default behavior (the
    // base class implementation) does just that.

    //virtual int pbackfail(int c = EOF);
    //virtual int underflow();

    // we don't do input so always return EOF
    virtual int uflow() {return EOF;}

    // we don't do input so always return 0 chars read
    virtual int xsgetn(char *, int) {return 0;}

    // Calls outputchar() for each character.
    virtual int xsputn(const char *s, int n);

    // We use the default behavior which returns a streampos that is in an invalid
    // position. We do not support repositioning on this stream.

    //virtual streampos seekoff(streamoff off,
```

```
    //                      ios::seekdir way,
    //                      ios::openmode which =
    //                          ios::in | ios::out);

    //virtual streampos seekpos(streampos sp,
    //                      ios::openmode which =
    //                          ios::in | ios::out);

    // We don't support setting the buffer so we use the default which is just to
    // return "this."

    //virtual streambuf *setbuf(char *s, int n);

    // There is nothing to sync with, so we just do the default which is to return
    // zero, indicating no error.

    // virtual int sync();

    // Actually, as an alternative implementation it would be possible to use this
    // function to call our soft flush routine. In practice, there would be no different
    // result. sync() is usually only called by pubsync(), which is usually only called
    // by flush(), which is usually only called by the endl manipulator function after
    // it has streamed '\n'. So the soft flush would always follow a hard flush and
    // result in a no-op.

  private:
    enum
    {
      kMaxDebugStrReadableString = 60,
      kSizeOfSemicolonG = 2,
      kSizeOfLengthByte = 1,
      kTabSize = 5,
      kStop = true
    };

    // the buffer
    char pbeg[ kSizeOfLengthByte +
          kMaxDebugStrReadableString +
          kSizeOfSemic olonG];

    // location of the next streamed char
    char *pnext;

    // This always points to the end of the readable string buffer. Once it is set in
    // init(), it is never modified
    char *pend;

    // the number of queued alerts
    int alertCount;

    void init();

    void outputchar(char c);
    void formfeed();
    void horztab();
    void backspace();
    void verttab();
    void alert();
    void addchar(char c);
    void softflush();
    void flushdebugstring(int stop = false);
    void flushalerts();

  friend class debugbufinit;
};

class debugbufinit
{
    static unsigned int count;
  public:
    debugbufinit();

    // Our destructor is not virtual. It is important that objects of this class have no
    // memory footprint. We will end up with one object of this class per translation
    // unit (.cp file). If this class has any virtual member functions then objects of
    // this class would have v tables in memory. Since this is not intended to be a
    // base class for other class, there is no need to be virtual.

    ~debugbufinit();
};
```

Of the `virtual` functions we inherited from `streambuf`, four are for input, which we don't support. For two, `pbackfail` and `underflow`, we can just accept the default behavior, and for the other two, `uflow` and `xsgetn`, we write trivial routines that return values which indicate that reading is not supported. Two other functions, `seekoff` and `seekpos`, are for positioning the stream pointer – another feature that we can't support. We also do not allow the caller to set our buffer, so `setbuf` is not supported. The `sync` function is also unneeded. The only routines that really do any work are `overflow`, which calls our `private` member `outputchar` once, and `xsputn`, which calls `outputchar` once for each character passed to it.

Our `private` section includes some constants defined as an `enum`, the buffer and the pointers that we need to manage it, a counter for buffering `SysBeep` calls, and our private functions. I'll discuss the private member functions later.

We finally return to the observation that instead of constructors and a destructor for `debugbuf`, a separate class, the `debugbufinit` class, is declared. This attempts to work around a problem with static objects. Before explaining the problem and what I've done about it, let me point out that I have not implemented a complete solution. *Do not stream to* `dout` *in the constructor of a static (or global) object* unless you are prepared for your application to crash. This is called "crossing the streams". See *Ghostbusters*.

`dout` is a static object and, like all static objects, its constructor will be called before the first line of `main` is executed. But the first line of `main` is not the first line of code that is executed. If a static object streams data to `dout` in its constructor, this code may be executed before `dout` is constructed.

There is no way to reliably order construction of static objects in different translation units (.cp files), but we do know that *within* translation units, static objects are constructed in the order in which they appear. That is why we have the class `debugbufinit` and why the header file `doutstream` declares a static object of this type. Note that it is not declared `extern`. Each translation unit that includes `doutstream` has its own object of type `debugbufinit` (which is why it is important that it does not have a memory footprint).

When the `debugbufinit` object is constructed, it uses its static member, `count`, to determine if it is the first object of its class to be constructed. If it is, then it calls the `init` member of the static `debugstream` object. The `init` member performs the function of a constructor for the `debugbuf` class. This way, `debugstream` gets constructed only once, at the time of the first construction of a `debugbufinit` object. Since `init` may be called before or after the real constructor is called, it is important that the constructor is a "no op".

P.J. Plauger explains this problem, along with the solution used in his implementation of the standard library streams (`cout`, `cin`, and `cerr`), in his book, *The Draft Standard C++ Library*, which I recommend. Close inspection reveals why my implementation is not a complete solution. Although I can

guarantee proper construction of the `debugstream` object, the same cannot be said for the `dout` object.

I suspect that a complete solution exists for both Metrowerks and for Symantec, but I do not believe that any single solution works for both. In any case, the complete solution is left as an exercise for the reader (I've always wanted to say that).

Member Function Definitions

As the listing for **debugbuf.cp** shows, the implementation of both `debugbuf` and `debugbufinit` is straightforward.

Listing 4: debugbuf.cp

Definitions of debugbuf and debugbufinit

The debugbuf class manages a buffer using a switch statement to differentiate between characters with special meanings.

```
#include "doutstream"

#include <string.h> // for strlen() and strcpy()
#include <OSUtils.h>// for SysBeep();

static debugbuf debugstream;
ostream dout(&debugstream);

unsigned int debugbufinit::count = 0;

// this is debug code - performance is not a goal

void debugbuf::init() // called by friend class debugbufinit
{
  pbeg[0] = '\0';
  pnext = pbeg + kSizeOfLengthByte;
  pend = pnext + kMaxDebugStrReadableString;
  alertCount = 0;
}


int debugbuf::overflow(int c)
{
  if (EOF == c)
  {
    return '\0'; // returning EOF indicates an error
  }
  else
  {
    outputchar(c);
    return c;
  }
}


int debugbuf::xsputn(const char *s, int n)
{
  for (int i = 0; i < n; ++i)
  {
    outputchar(s[i]);
  }
  return n;   // we always process all of the chars
}


void debugbuf::outputchar(char c)
{
  switch (c)
  {
    case '\b':
      backspace();
      break;
    case '\f':
      formfeed();
      break;
    case '\n':
      flushdebugstring();
      break;
    case '\r':
      softflush();
      break;
    case '\t':
      horztab();
      break;
    case '\v':
      verttab();
      break;
    case '\a':
      alert();
      break;
    case '\0':
    case '\x03':
      flushdebugstring(kStop);
      break;
    default:
      addchar(c);
      break;
  }
}


void debugbuf::backspace()
{
  if (pbeg[0])  // if the buffer is empty, don't bother
  {
    --pnext;
    --pbeg[0];
  }
  // note that alerts cannot be backspaced away - a possible enhancement
}


void debugbuf::formfeed()
{
  softflush();
  strcpy(pbeg, (char *)
    "\p_____________________________________________");
  pnext += strlen(pnext);
  flushdebugstring();
}


// both horizontal and verticle tabbing is done by brute
// force - performance is not a goal
void debugbuf::horztab()
{
  // we don't wrap tabs so if we are within kTabSize of
  // the end of the buffer then we just flush
  if (pend - pnext <= kTabSize)
  {
    flushdebugstring();
  }
  else
  {
    for (int i = 0; i < kTabSize; ++i)
    {
      addchar(' ');
    }
  }
}


void debugbuf::verttab()
{
  int position = pnext - &pbeg[1];
  flushdebugstring();
  for (int i = 0; i < position; ++i)
  {
    addchar(' ');
  }
}


void debugbuf::alert()
{
  ++alertCount;
}
```

```
void debugbuf::addchar(char c)
{
  *pnext++ = c;
  ++pbeg[0];
  if (pnext == pend)
  {
    flushdebugstring();
  }
}


void debugbuf::softflush()
{
  if ('\0' != pbeg[0])
  {
    flushdebugstring();
  }
  else
  {
    flushalerts();
  }
}

void debugbuf::flushdebugstring(int stop)
{
  if (!stop)
  {
    addchar(';');
    addchar('g');
  }
  flushalerts();
  DebugStr((unsigned char *)pbeg);
  pbeg[0] = '\0';
  pnext = &pbeg[1];
}
```

```
void debugbuf::flushalerts()
{
  while (alertCount)
  {
    --alertCount;
    SysBeep(30);
  }
}

debugbufinit::debugbufinit()
{
  if (0 == count++)
  {
    debugstream.init();
  }
}

debugbufinit::~debugbufinit()
{
  if (0 == --count)
  {
    // nothing to dispose, but we should flush
    debugstream.softflush();
  }
}
```

The main entry points are the protected members `overflow` and `xsputn`, both of which call `outputchar`. `outputchar` uses a `switch` statement to call one the following if the character is "special": `flushdebugstring`, `softflush`, `formfeed`, `horztab`, `backspace`, `verttab`, or `alert`. If the character is "normal" we call `addchar`.

- If the character being processed is `doutbackspace` (`'\b'`), we reduce the length of the Pascal string in the buffer by one and back up the pointer by one character. Of course, we check first to be certain that there is at least one character in the buffer.

- If the character being processed is `doutformfeed` (`'\f'`), we flush any characters already in the buffer, fill the buffer with underscores, then flush the underscores.

- If the character being processed is a horizontal tab or `douttab` (`'\t'`), we treat it as if it were `kTabSize` (5) spaces, except that we don't wrap remaining spaces to the next line. We don't support tab stops.

- If the character being processed is a vertical tab or `doutverttab` (`'\v'`), we calculate how many characters are currently in the buffer, flush the buffer, and add a space character for each character that had been in the buffer.

- If the character being processed is an alert (`'\a'`), we increment the alert counter. Since all alerts are the same (just a call to `SysBeep`), we can "buffer" them with just a counter.

- If the character being processed is "normal", we process it in `addchar`. The new character is added to the end of the Pascal string in our buffer. When the string fills the readable string buffer, we flush the buffer by calling `flushdebugstring`.

Notice that I said "the readable string buffer". The buffer is actually two bytes larger than the largest Pascal string that we can handle. This is to save room to append a semicolon and the letter "g".

Usually when `DebugStr` is called, MacsBug displays the Pascal string that is passed to it and waits for commands from the user. This is not really the behavior that we want. We want the string stored in the MacsBug buffer, but we don't want to stop the application and drop into MacsBug every time data is streamed to `dout`. To work around this, we take advantage of the `DebugStr`/MacsBug command processing feature. When MacsBug receives the string passed to it by `DebugStr`, it displays the contents of the string up to, but not including, the first semicolon (if there is one). Any characters after the semicolon are treated as a command for MacsBug to execute.

By appending "`;g`" to the string passed in `DebugStr`, we cause MacsBug to execute the "g" or "go" command which resumes execution of the suspended application. Since we are going to append this to almost every call to `DebugStr` (the exceptions being when `'\0'` or EOT are streamed), we never let the Pascal string in the buffer to grow into the last two bytes of the buffer.

The `softflush` routine is called when the user streams `doutsoftflush` (`'\r'`). We check to see if there are any characters in the buffer. If there are, we call `flushdebugstring`, and if there aren't, we call `flushalerts`. Since `flushdebugstring` calls `flushalerts`, alerts are always flushed.

`flushalerts` calls `SysBeep` the number of times specified in `alertCount` and resets `alertCount` to zero.

The routine that actually calls `DebugStr` is `flushdebugstring`, which is called when the user streams `'\n'`, `'\0'`, or EOT. `flushdebugstring` takes a single parameter which is used to determine whether or not to append "`;g`" to the buffer before passing it to `DebugStr`. After appending the "`;g`" or not, we flush alerts with a call to `flushalerts`, call `DebugStr`, and then zero out the string in the buffer.

Bibliography and References

Information on using the standard C++ iostreams library is readily available. Any recently published work on ANSI C++ would include information on library usage. Implementation is a somewhat different matter. The `dout` Library would probably not have been possible without:

Plauger, P.J. *The Draft Standard C++ Library.* Prentice Hall, 1995.

For information on MacsBug:

Apple Computer. *MacsBug Reference and Debugging Guide.* Addison Wesley, 1990.

NEWSBITS

By Will Iverson

General Magic Sets its Cap at the Internet

On July 11th, General Magic announced Presto!Mail and Presto!Link, a Web browser and email application for Magic Cap. On August 30th, the ill-conceived PersonaLink service will cease, hopefully bringing an end to the proprietary nature of Magic Cap. Now that General Magic has discovered TCP/IP, PPP, POP and SMTP, we can only hope that MagicCap fares better in the marketplace.

This is actually a great move for General Magic and the PDA market in general. Both the Magic Cap and Newton OS featured brilliant software and hardware designs, but they were both completely incompatible with everything else anyone owned. This might have worked with a razor-blade strategy (sell the razor cheap, clean up on the blades), but with such a high entry cost, few bit onto the technology. By adhering to standards, these PDAs justify their cost without forcing users to absorb propietary formats. With luck, this will spark a renaissance of the PDA market (and perhaps get some return on that investment money).

General Magic: http://www.genmagic.com/

Paragraph Puts Up a VRML 2.0 Web Page

VRML enthusiasts will get a kick out of the VRML Power Friday Resource Page. Featuring transcripts and papers from the likes of Apple, IBM, Netscape, and SGI, it provides a great shot of the current state of the mind on this still nascent technology. Virtual reality is the promise, and VRML seems to be the path. You'll know that 3D is really coming along when popular rendering programs export reasonably-sized VRML files.

VRML Power Friday Resource Page:
http://www.paragraph.com/960703/vrml_power_friday/

VOODOO Announces a New Version

UNI SOFTWARE PLUS has announced VOODOO 1.7, featuring Macintosh Drag and Drop, local file locking, printing of the project history, and "an assortment of other usability and performance enhancements, as well as some minor bug fixes".

For a product description, see "Keeping Things Straight, Orthogonally" (*MacTech Magazine* 12.6 [June 1996] 61–70).

UNI SOFTWARE VOODOO:
http://www.unisoft.co.at/e/products/voodoo.html

Game Sprockets Goes Gold

On July 2nd, Apple announced the GM 1.0 version of Game Sprockets (with new versions slated to appear any time now). This excellent package features six primary components: NetSprocket, SoundSprocket, SpeechSprocket (speech recognition), InputSprocket, DrawSprocket, and QuickDraw 3D RAVE (a 3D API at a lower level than normal QuickDraw 3D). Each component provides higher-level access to these common tasks (with the exception of QuickDraw 3D RAVE, which provides a lower-level, performance-intensive API). DrawSprocket, for example, allows a developer to put together quickly a double- or even triple-buffering scheme which automatically takes advantage of any hardware support. By placing the burden of such things as worrying about what Performa supports which hardware on the Game Sprocket library, developers are free to concentrate on the game itself.

All of this does not come for free. The Game Sprockets are implemented as PowerPC shared libraries only; 68K developers and players are left in the lurch. Duplicating the functionality of the Sprockets yourself for the 68K completely destroys the point of using the Game Sprockets. Eliminating the 68K cycle is an excellent way to keep development costs down; it is up to the developer to decide if this makes sense financially. Among those who have decided that this is a price worth paying are Future Point (developing the forthcoming Warcraft 2), MacPlay, Bungie, and Wirehead Systems.

Although they are designed for use with games, certain Game Sprockets are decidedly useful in other applications as well. DrawSprocket is useful whenever high-speed animation is required, and NetSprocket is an excellent tool for general lightweight networking development.

The Game Sprockets do, however, raise the spectre of namespace collisions, which are always possible with shared libraries. This is not to say that you will have this problem, but rather to call attention to the inelegant, brute-force manner in which the problem is dealt with. With such function names as `ISpElement_NewVirtualFromNeeds`, it is readily apparent what library is being called (once you figure the scheme out); but it is also rather messy. Namespaces would presumably be a step toward resolving this, but at the time of this writing none of the currently shipping Macintosh compilers support them.

Apropos of nothing at all, readers might like to bear in mind that Sprocket™ is a trademark of Xplain Corporation, publishers of *MacTech Magazine*.

Game Sprockets (don't try this with a monochrome monitor):
http://dev.info.apple.com/evangelism/games/games.html

Web Broadcasting Talks to FileMaker Pro...

On June 24th, Web Broadcasting announced support for FileMaker Pro 3.0, allowing you to easily publish databases on the Web. For intranet users, this may be the ideal way to develop client apps for inhouse production. This new version includes support for FileMaker Pro relational and portal fields, support for Infoseek-like "Next xx" records when performing a search, capability to mix "AND" and "OR" searches, and the capability to capture all passed http parameters (exceedingly useful for logging use).

Trial version of WEB FM 2.0v3:
http://macweb.com/webfm/

Current registered users of WEB FM 2.0 can download version 3 at:
http://macweb.com/webfm/airroom/live.html

...And Tango Dances With It

Hot on the heels of Web Broadcasting, EveryWare shipped Tango for FileMaker Pro 3.0 on July 1st. EveryWare emphasized the performance and drag-and-drop ease of development for this new version of Tango/FM. Although significantly more expensive than Web Broadcasting's solution ($395 versus $195), Tango provides a better upward migration path (including another EveryWare product, the Butler SQL – read on), and appears to be more popular.

Tango for FileMaker Pro 3.0:
http://www.everyware.com/tangofm/

Butler SQL 2.0

EveryWare announced on May 13th Butler SQL 2.0, which now supports TCP/IP, Alter Table statements, ODBC AppleScript, default ODBC and DAL stored procedures, and tracing ODBC commands from client applications. Butler SQL now comes bundled with Sterling Software's CLEAR:Access, a drag-and-drop SQL reporting tool. EveryWare claims that ODBC support gives users a three to four times speed improvement over version 1.5. The product includes four toolkits to access the database, including a C/C++ framework, an AppleScript package (which actually supports any OSA language, despite the name), an XCMD toolkit, and FirstClass integration.

Butler:
http://www.everyware.com/WWW_info_pages/Butler/Butler_SQL.html

Nisus Revives QUED/M with 3.0

Nisus is back with QUED/M 3.0, which boasts features such as unlimited undo and redo, syntax coloring, GREP, noncontiguous selections, and "Macro Programming Dialect" (users of the Nisus Writer word processor will know what this is). It includes macros for HTML authoring, and a builder which allows you to add macros directly to the menu structure. It is now PowerPC-native, supports Apple Drag and Drop, and supports both Symantec and Metrowerks development tools. Perhaps most interestingly, it supports "text folding", allowing a user to treat code as an outline, collapsing and expanding comments and loops (a feature supported for years by Frontier). In addition, it comes with Celestin Company's Apprentice 4 source code CD. All this for only $69. Can you say "price war"?

QUED/M information and free demo:
http://www.nisus-soft.com/QUED_M.html

VIP BASIC 2.0

Mainstay announced on May 8th the arrival of a new version of their BASIC tools. The Macintosh has long lacked a competitor to Visual Basic on the PC, and Mainstay looks to be positioning VIP BASIC 2.0 as such a product (to judge by their advertising). Featuring a forms-based visual development tool, Mainstay claims that users can export at any time to straight C (fully compatible with Metrowerks CodeWarrior) for the fastest possible executables. The tool includes a very small database which allows up to 32K of data; for larger databases, the user is required to purchase a separate VIP-BASIC Database Manager product. VIP-BASIC allows direct access to the Macintosh Toolbox, includes a framework, and generates PowerPC-native applications.

Mainstay VIP BASIC:
http://www.mstay.com/product_page.html

Tenon Revs MachTen: Price Subtraction, Ada Addition

The folks at Tenon have been busy developing software, and are getting much more aggressive with their marketing to boot. MachTen is a full-featured BSD UNIX for Macintosh; unlike Linux, however, it doesn't take over your computer in the process. Rather, much like SoftWindows, it runs in a window on your Macintosh. The applications which run inside this shell are fully protected and preemptive. As of June 14th, the package includes NCSA httpd, DNS, POP3, IP forwarding, a complete UNIX-based C/C++ development, a high-performance X server, a complete X11R6 client development environment, and a Motif toolkit. The application is fully Macintosh-savvy, and supports mounting AppleShare volumes under NFS. As if that weren't enough, a bundled version of BBEdit is included.

In addition, on June 24th, Tenon announced the availability of GNAT-Ada. Using the gcc back-end, combined with the GNAT-Ada front-end, Tenon has brought Ada 95 to the Macintosh and Power Macintosh. While not complete as of this writing, the package is available as a free update.

Both the 68K and PowerPC versions of MachTen are available now at a suggested price of $695 ($495 for the "personal" 68K version).

Tenon MachTen: http://www.tenon.com/
GNAT-Ada: http://gnat-mac.com/macada/

Blaze Tech Debugs Apple's Debugging Course

Blaze Technology's Malcolm Teas has just finished a revision of Apple Developer University's course called "Debugging Skills and Strategies". This revision provides a much needed update, to address PowerPC debugging and new debugging tools. If you are interested in learning more about low-level debugging, including such valuable tidbits as low-level knowledge of memory layouts, stack frames, and cross-fragment calls, you should absolutely check this one out. To cover the additional material, the class has been expanded to four days instead of the earlier three.

Apple Developer University: http://dev.info.apple.com/du.html
Blaze Technology: http://www.btech.com/

Altura Announces QuickHelp 4.0

Altura announced QuickHelp 4.0 on July 1, a tool for electronic indexed documentation. QuickHelp 4.0 adds support for the Windows95 version of WinHelp, including compatibility with Microsoft WinHelp 4.0 source.

Altura: http://www.altura.com

Quick Off the Mark

On July 2, Kelly Rowan Mark was born, at 8 lbs., 3 oz. Dave Mark, the proud poppa, announces that the beautiful baby girl and mother are both doing fine. While *MacTech Magazine* does not recommend the use of tobacco products, a round of virtual cigars is certainly appropriate. Rumor has it the young lass takes after her dad and is already babbling PowerPC opcodes as she naps.

Important Reminder

Information for Newsbits should be submitted via email, and sent to press_releases@mactech.com (not to any of the other MacTech email addresses, please!).

Also, don't forget our moderated news mailing list, MacDev-1™, which is distributed to thousands. To a submit a posting to MacDev-1, just send it as email to MacDev-1@listmail.xplain.com. (To become a recipient of MacDev-1 mailings, send mail to listserv@listmail.xplain.com, with "`subscribe MacDev-1`" as the body of the message.)

DIALOGUE BOX

By Matt Neuburg, letters@mactech.com

Hello Java, Good-bye OLE and OpenDoc

Sometimes, when we are lucky, reality sneaks up and screams "Hello!" when we most need it.

Thinking about how to write extensible applications in Java, I was listing requirements, procedures, and issues for communication and control between the main application and the extensions. What is the minimum core of OLE and OpenDoc? Does it really have to be as complicated as OpenDoc?

Then reality suddenly screamed, "Hello! Isn't a Java-enabled browser one of the most extensible applications we have ever seen? How is that done?"

A browser can load and run essentially "anonymous" applets, of any complexity, that can do just about anything – even bring up windows and work with files (when permitted). What more could we want?

Well, for extensible applications in general, we could want custom menus, closer integration into the application, and more access to the application's data. As it happens, all this is possible – right now – with Java; the implementation of a browser shows us how.

We extend the Applet class to make our own applets. The Applet class establishes a few "entry points" (methods) through which a browser can tell the applet to initialize itself, run, update, and so on. The applet can do whatever else you want in addition to these few responsibilities.

For its part, the browser must implement the AppletContext interface, which dictates the services the browser must provide to any applet. If both sides fulfill these contracts, the browser can use Java's ClassLoader to load an applet from a local or remote file, place it into the browser's own window, and run it.

To implement our own extensible applications, we don't have to use applets (although we certainly can); we just need to set up the kind of contracts established by the Applet class and the AppletContext interface. Perhaps we will include ways to share menus and application data; the particular methods are our call (so to speak).

Now think of the silliness of first implementing a Java-enabled OpenDoc part, then rewriting applications to accept OpenDoc parts! Or just rewriting for OpenDoc in general. Why not rewrite the applications in Java, and get straight to the point?

"Hello!" Which is the *actually existing revolution* – Java, or OLE/OpenDoc?

It's time to tell OLE and OpenDoc (and VBXs, OCXs, XFCNs, MOBJs, XTNDs, and so on) "Good-bye!"

"Uncle Dave" Moffat

Mac Web Tools Redux

In the May 1996 *Dialogue Box*, Ziya Oz writes:

"At the last Macworld Expo in Boston, I asked myself: If Ceneca was not formed by ex-Taligent people, would they have developed Page/SiteMill for the Mac OS first? Would they even have developed for the Mac at all? ... My answers are: No and No again."

His conclusions are erroneous. Ceneca initially planned to write PageMill for Windows. However, when they spoke to Web authors at several companies, they found out that Macintosh was the dominant platform for creating Web content. Ceneca therefore changed their strategy and developed a Mac version instead. (I learned this from Robert Seidl of Ceneca/Adobe at the WWDC.)

The majority of Web content is created on Macs. Macintosh is also the second most popular Web server, after UNIX. A developer writing content tools would be wise to publish a Mac version of his product.

Terry Morse

Language Wars Redux

Many Mac OS programmers still code in Pascal – and that's OK. But they (and *MacTech Magazine*, too) should give another language a try. No, I'm not talking about C++ or Java; I'm talking about Oberon-2 (Pascal's grandchild, so to speak). Interested? Then read on.

MacOberon, Oberon System 3, and Oberon/F generate 68K code. PowerOberon generates PowerPC code, but has no linker yet. Or you can use MPW. The URLs are:

ftp://ftp.inf.ethz.ch/pub/Oberon/
ftp://oberon.ssw.uni-linz.ac.at/pub/Oberon/PowerMac/
http://frodo.informatik.uni-ulm.de/projekte/MPWOberon/

As a CodeWarrior, I hope that Metrowerks will integrate this fantastic language into their IDE. But, till then, I'm perfectly happy with C++ and PowerPlant.

By the way, to "serious" Oberon users I recommend *Programming in Oberon* and *Object-oriented Programming in Oberon-2* (both from Addison-Wesley).

Paul E. Sevinc

[*Our paucity of articles written in Oberon is not due to any want of trying. In fact, we have a review of Oberon/F in the works; watch for it in a future issue. As always, the best way to ensure coverage of your favorite area of expertise is to participate by writing us an article yourself – man*]

TIPS & TIDBITS

By Steve Sisak

TIP OF THE MONTH

DID YOU TAKE OUT THE GARBAGE?

A somewhat obscure but useful way of determing whether objects are getting deleted in C++ is to do this:

1. In the class implementation (usually in the header file), add a destructor:

```
class MyClass
{
public:
   virtual ~MyClass();
```

2. *Don't* implement the destructor; i.e., don't write the actual code for the destructor method.

3. Run the project. If you get link errors for the destructors, this means that the destructors are getting called (i.e. the objects are getting deleted), which is what you want. If you don't get a link error, this means that the object never gets deleted, which could signify a possible memory leak.

Jeremy Vineyard

PRE-FLIGHTING EDITTEXT

For a program I recently wrote, I wanted to do user input checking for a dialog box as the user was typing into an EditText field. I wrote a routine that figures out what the contents of a specific EditText box will be if the current event is allowed to be processed. I then called the routine from my custom dialog event proc, and analyzed the results. That way, I could easily decide if the user's action would produce valid results, or if I needed to abort the current event.

`IsDlgControl` is a simple check for control characters we always want to process. `DivineNewItemString` is the routine that does all the work. `PStrCopy` is a little routine for copying Pascal strings; I'm sure other people have better ways of doing this.

Michael Trent

```
// some control key constants.
#define kEnterKey               3
#define kBackspace              8
#define kTab                    9
#define kReturnKey              13
#define kEscapeKey              27
#define kLeftKey                28
#define kRightKey               29
#define kUpKey                  30
#define kDownKey                31
#define kDelete                 0xFF

/* IsDlgControl
 * Returns true if c is a special control key (say an arrow key, or escape).
 * Otherwise returns false.
 */
Boolean IsDlgControl(char c)
{
        if ((c >= kEscapeKey) && (c <= kDownKey)) return true;
        if ((c == kReturnKey) || (c == kEnterKey)
        || (c == kDelete) || (c == kBackspace) || (c == kTab))
                return true;

        return false;
}

/* PStrCpy
 * A little routine to copy Pascal strings. Provided for those people who don't
 * already use BlockMove() to do this for them ...
 */
void PStrCpy(Str255 s, const Str255 t)
{
        short i;

        for (i=0; i<=s[0]; i++) {
                s[i]=t[i];
        }
}

/* DivineNewItemString
 * Given a DialogPtr, EventPtr and an item number for the active EditText DLOG
 * Item, it returns what the string will be if the current event is processed.
 * It should be called from a custom dialog event proc.
 */
void DivineNewItemString (
     DialogPtr d, EventRecord *e, short item, Str255 output)
{
        short           *TEScrpLength = (short *)0x0AB0;
        DialogRecord    *dr;
        TEHandle        teh;
        char            c;
        Str255          input, text;
        short           selStart, selEnd;
        short           i;
        short           outStrIdx=0;
```

*Send us your tips or we'll install EvenBetterBusError on your machine! On the other hand, we might just pay you **$25** for each tip we use, or **$50** for Tip of the Month. You can take your award in goods, subscriptions or US$. Make sure any code compiles, and send tips (and where to mail your winnings) to our **new Tips e-mail address** at tips@mactech.com. (See page 2 for our other addresses.)*

```
        short           iType;
        Handle          iHandle;
        Rect            iRect;

        // get the text string
        GetDItem(d,item,&iType,&iHandle,&iRect);
        GetIText(iHandle,text);

       // Set the input string
        c = (e->message & charCodeMask);

        if (IsDlgControl(c)) {  // if it's a control char, return the
                                // item's text.
                PStrCpy(output,text);
                return;
        } else if (e->modifiers & cmdKey) {
                if  ((c == 'v') || (c == 'V')) {
                // if pasting, get the pasted string.
                        (void)TEFromScrap();
                        HLock(iHandle);
                        iHandle = TEScrapHandle();
                        for (i=0; i< *TEScrpLength; i++) {
                                input[i+1]=((unsigned char *)
                                        (*iHandle))[i];
                        }
                        input[0]=*TEScrpLength;
                        HUnlock(iHandle);
                } else {  // if any other command stroke
                        PStrCpy(output,text);
                        return;
                }
        } else {
        // else, set the input string equal to the new character
                input[0]=1;
                input[1]=(unsigned char) c;
        }

        // get the selection point from the TERec
        dr = (DialogRecord *)d;
        teh = dr->textH;
        selStart=(*teh)->selStart;
        selEnd=(*teh)->selEnd;

        // generate output string:
        //  copy the first bit of text
        for (i=1; i<=selStart; i++) {
                output[++outStrIdx]=text[i];
        }
        //  copy the input string
        for (i=1; i<=input[0]; i++) {
                output[++outStrIdx]=input[i];
        }
        //  copy the last part of text
        for (i=selEnd+1; i<=text[0]; i++) {
                output[++outStrIdx]=text[i];
        }
        //  lastly, set the length
        output[0] = outStrIdx;
}
```

A Free Tool From Apple

You can use the Finder's "About this Macintosh" window to see the size your application's heap, and the amount of free memory that is available in your heap. This information is only updated during idle time, though, so don't count on it to always be accurate. [*Here's the real tip: use Balloon Help to get a numeric representation of the information. – sgs*]

David Lawrence

Avoiding the Menu Item Gotcha

Watch out when using AppendMenu or InsertMenuItem. I'm using AppendMenu to create a popup menu based on user input. Since AppendMenu interprets certain characters as menu attributes (such as command-key equivalents, menu dimming, etc.), if a user enters one of these characters, you will run into a problem with your menu. To avoid this, call AppendMenu or InsertMenuItem with a dummy string such as `"\p "` (the string must not be empty), and then call SetMenuItemText using the real string, like so:

```
Str255 theUserName;

// Get the string to display in the menu
theUser->GetUserName (theUserName);

// Append an empty string to the menu
AppendMenu (theMenuHandle, "p\ ");

// Change the menu item to the userName
SetMenuItemText (theMenuHandle, theIndex, theUserName);
```

Tim Pedone

World's Most Original Use of ResEdit

If you are looking in the code of a PICT, you can often see a color table with 256 colors even if you use only 2 or 3 colors. This can represent a massive waste of disk space. I tried to find an app to compact the color table but I couldn't find one – until finally I found ResEdit!

My trick: create a `'cicn'` resource in ResEdit, paste in your PICT or draw it, select the part for using in the PICT, copy and paste it into an actual PICT. The only limitation is the maximum size: 64 by 64.

Henri Clerc

No, Wait – World's Most Truly Totally Original Use Of ResEdit

This might be one everybody knows, but I just figured it out and it's saved me a load of time, so here goes.

ResEdit is a Drag-and-Drop application!!! This works for *any* type of file. No longer must eons be frittered away navigating through countless folders in open dialog boxes to access resource forks of files hidden deep within the murky depths of your hard drive. Just do a **Find** in the Finder and drag it to ResEdit – I keep an alias on my desktop. Happy hacking!

Joshua "Vampiric Bunny" Glazer

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

LIST OF PRODUCTS


Mac OS
Scripter
The ultimate authoring and development environment for AppleScript
Danny Goodman's
Apple Guide
Starter Kit
Danny Goodman
Jeremy Joan Hewes
FRAGMEN
OF YOUR
IMAGINAT
Code Fragme
Resources for Pow
and Macintosh
Joe
SIC for the
ewton
John Schettino · Liz O'Hara
New Technology Building Blocks
Macintosh
Programming
Techniques
A Foundation for All Macintosh Programmers
Learn essential techniques for expert Macintosh programming
Understand terms and concepts unique to Macintosh programming
Gain hands-on experience with interactive software tutorial
Dan Parks Sydow
LEARN
C
ON THE
MACINTOSH
SECOND EDITION
SUBSCRIPTION
VERSION
SYMANTEC.
C++
DEVELOPMENT SYSTEM FOR
POWER MACINTOSH
WARRIOR
RONZE 8
CodeWarrior
FACESPAN
The Interface Designer and Application Builder.
Dave Mark Cartwright Reed
Macintosh C
Programming
PRIMER
Volume I
Second Edition
Inside the Toolbox Using THINK C
OPTIMIZ
Power
CODE
Gary Kacmar
leScript
Additions Guide
Essential
Developer
DEPOT!

Open Programmer's Hours...

...and all day too. In fact, we're open 24 hours a day, 365 days a year!

Which means that whenever you get a craving for the latest in developer tools, we're just a couple of keystrokes away.

You can browse for hours (we don't mind), pick up the items you want, and conveniently pay with your major credit card at our virtual check-out. Then, kick back and wait for your goodies to arrive - all without ever leaving your house!

Best of all you'll shop confidently, because at Developer Depot, you're guaranteed complete satisfaction and the best prices for 30 days after your purchase.

Talk about a safety net!

That's right, and speaking of nets, get connected and come check out the vastest on-line selection of developer products... any day, any time!

Developer DEPOT

http://www.devdepot.com

Dear Developer,

Since March, 1985, **MacTech**™ Magazine (then MacTutor) has been the place for you to find Macintosh Development products – the majors as well as those harder to find. Back then … and today, our focus has always been for the Macintosh developer community to grow and prosper.

But you deserve more than simple listings – and you've asked to make it easier to find and order products while maintaining our renowned customer service and pricing. We've been listening …

We are therefore proud to announce **Developer Depot**™ – a new mail order concept and entity dedicated to you, the Mac OS Developer. What makes Developer Depot so good?

- World renowned customer service
- 24-hour and full accessibility via our continually updated web site (http://www.devdepot.com)
- Satisfaction and best price guaranteed
- A new, easy to remember toll free phone number (**800-MACDEV-1**)
- Great selection (hundreds of products)
- Convenience: you can order in many ways (web site, e-mail, fax, phone)
- A new full color, organized, and expanded catalog – updated monthly

Additionally, we are a Macintosh house using an all Macintosh system. Our knowledgeable staff can answer your questions about products – and if they can't, they'll find the answers for you.

While Developer Depot is completely separate from MacTech Magazine, Developer Depot is the primary source for all MacTech products.

Let us know what you think – we're here to serve you.
Developer Depot: The products you need with the service and prices you deserve.

Neil Ticktin
Chief Executive Officer

Andrea J. Sniderman
Chief Operating Officer

TABLE OF CONTENTS

Developer Depot 30 day Money Back, Price and Satisfaction Guarantee
Developer Depot products are sold with a 30 Day money back guarantee on user satisfaction, lower prices and against defects. If, for any reason, you are not satisfied or find the same product at a lower price within 30 days, please call Customer Service at 800-MACDEV-1 and request a Return Merchandise Authorization (RMA) number to get a full refund or the difference in price (where applicable). You must return undamaged product at your expense, including all its original packaging, documentation and the blank warranty card if applicable. Developer Depot will replace defective product upon receipt of the defective merchandise. Please remember to back up your data before installation of any new hardware, software, or peripherals; we cannot be responsible for any lost data. Policies, item availability, and prices are subject to change without notice. The price in effect when we receive your order will be the price that you are charged. We are not responsible for any typographical errors in this or any other catalog, nor for any misstatements from any vendor. Purchase orders are also accepted but must be in writing, signed, come with a contact person and a telephone number, and mailed to P.O. Box 5200, Westlake Village, CA 91359-5200. Faxed copies and purchase order numbers alone are not acceptable.

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

MacTech Magazine

Subscriptions: US magazine:
$47 for 12 issues (MTYRDM)
Canadian: **$59** for 12 issues (MTYRCM)
International: **$97** for 12 issues (MTYRFM)
Back Issues: **$5** each (subject to availability)

MacTech CD-ROM Volumes 1-11 NEW!

- 1420+ articles, from all 127 issues of MacTech Magazine (1984-1995).
- Improved hypertext, and a new THINK Reference Viewer — for lightning quick access!
- New hyperlinks between articles allow you to jump to other relevant articles.
- 100+ MB of source code — use them in your own applications, with no royalties!
- Full version of THINK Reference™ 2.0. The original online guide to Inside Macintosh, Vols. I-VI.
- 80MB of FrameWorks/SFA archives. The most complete set of FrameWorks archives known.
- Sprocket™! MacTech's Tiny Framework that compiles quickly and supports System 7.5 features.
- The best threads from the Macintosh programmer newsgroups plus thousands of notes, tips, snippets, and gotchas.
- Popular tools that Macintosh programmers use to increase their productivity and much more!

Our Price **$89.00** (SMTCD11) Upgrades: Our Price **$39.00** (SMTCD11U)

NEW!

MacTech Mouse Pad

Slide on this! With an extra-large surface (11" by 10") and a deluxe sleek plastic coating, you'll be zooming across your screen in no time at all.
Speed limit not enforced!
Our Price **$8.95** (AMTPAD)

Order Toll-free
800-MACDEV-1
(800-622-3381)

EXCLUSIVES!

FrameWorks Magazine

$8 per back-issue, subject to availability. (MTFWBACK)

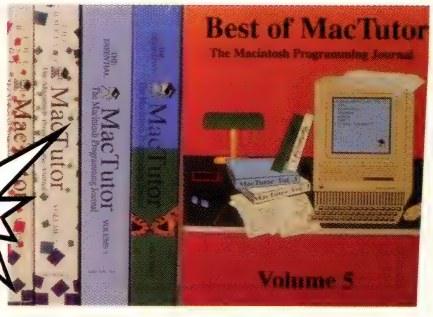
FrameWorks Source Code Disk

$10 per back issue, subject to availability. (MTFWSC)

MADACON '93 CD-ROM

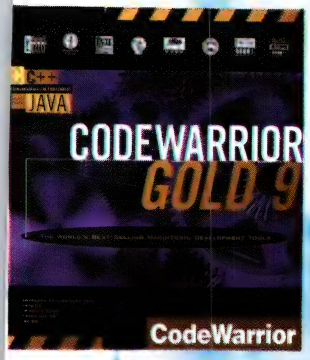
The highlights of MADACON '93, including Mike Potel on Pink, Bedrock, MacApp, OODLs, and more. Slides, articles, demos, audio, and QuickTime.
List $95.00 Our Price **$9.95** per set (SMADA93).

BLOW OUT!



Best of MacTutor

The Best of MacTutor Collection, Vols.3-5.
List $99.00 Our Price **$9.95** per set (BMTBEST).



CodeWarrior 9 Gold by Metrowerks

- Build applications for the Mac OS, Windows 95, Windows NT, Magic Cap, & the new Be OS.
- Includes C, C++ and Object Pascal compilers, source-level debuggers, object-oriented frameworks (PowerPlant, MacApp, MFC), Apple's MPW, complete online documentation and source code examples for all languages and platforms.
- The IDE software has been localized in eight languages plus English. CodeWarrior is sold on an annual subscription basis and has three major releases each year.

Our Price **$399.00** (SCWGOLD)

SEE RELATED CATEGORY:
Internet Related

SEE RELATED PRODUCTS:
• Code Manager • Inside Power Plant • Inside CodeWarrior 9 • C++ Programming with CodeWarrior • PowerMac Programming Starter Kit • Discover Programming for Macintosh • Learn C on the Macintosh • Metrowerks CodeWarrior Programming

CodeWarrior 8 Bronze

- Build applications for the Mac OS on 68K machines.
- Includes C, C++ and Object Pascal compilers, source-level debuggers, object-oriented frameworks (PowerPlant, MacApp), Apple's MPW, complete onlinedocumentation and source code examples for all languages and platforms.
- The IDE software has been localized in eight languages plus English.
- Sold on an annual subscription basis and has three major releases each year.

Our Price **$149.00** (SCWBRON)

CodeWarrior Mouse Pads

Fight your way through the night with these cool-looking pads!

- Arnold: Our Price **$8.95** (ACWPADA)
- Gears (not pictured): Our Price **$9.95** (AMGEAR)

CodeWarrior Wear (XL only)

You live it, you breath it... you might as well wear it!

- Black CodeWarrior Sweatshirt: Our Price **$29.95** (ACWSWEAT)
- "Blood, Sweat & Code" black long-sleeve shirt: Our Price **$14.95** (ACWBLOOD)
- Cross Platform white shirt (not pictured): Our Price **$14.95** (ACWXPS)
- Hawaii Five-0 shirt (not pictured): Our Price **$7.95** (ACWHAWAII)
- Arnold at work T-shirt (not pictured): Our Price **$9.95** (ACWARNLD)
- Hat (black or white): Our Price **$14.95** (ACWHAT)
- Winter Hat (not pictured): Our Price **$14.95** (AWINHAT)

Presenting Magic Cap, A Guide to General Magic's Revolutionary Communicator Software

- Perfect for novices as well as experts who want to take full advantage of Magic Cap.
- Step through its screens, see how it works, and learn ways to use it.
- Describes the power of smart messaging to customize the way you handle e- mail, a name card file that learns how to keep track of your contacts, a datebook that performs like a faithful assistant, and countless other abilities.

List $16.95 Our Price **$15.25** (BPRESMAGIC)

SmalltalkAgents® by Quasar Knowledge Systems

• Features a new generation of the Smalltalk language, QKS Smalltalk™. • Productivity is no longer measured in lines of code, but in project completion time. • Allows "live" direct manipulation of your objects. • Provides a dynamic, interactive and iterative development process. • Provides easy and full access to the features of the Mac OS™ and Mac Toolbox. • Link your non-Smalltalk code resources using our External Code Linking Toolkit™ (ECLT). • A sophisticated database for source code management provides an almost infinite variety of ways to cross-reference, access, view, and manipulate your code and objects. • Includes an Application Delivery Toolkit™ (ADT) that allows you to create royalty-free, stand-alone, double-clickable applications in just a matter of minutes. • Any component built in AO/S will function as an OpenDoc component or container and components from non-AO/S sources can be seamlessly integrated into the AO/S system.

Our Price **$695.00** (SSTA)

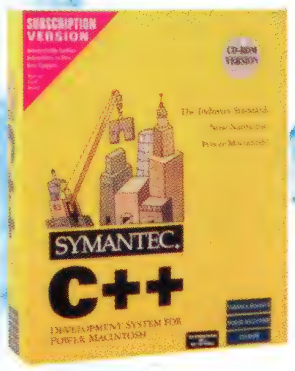
SEE RELATED CATEGORY:
Internet Related

SEE RELATED PRODUCTS: Symantec C++ Programming
SEE RELATED PRODUCTS: Learn C++ on the Macintosh
SEE RELATED PRODUCTS: Programming in Symantec C++
SEE RELATED PRODUCTS: Mastering theTHINK Class Library

Symantec C++ 8.5

- Native for Power Macintosh.
- Develop full-featured Power Macintosh applications quickly and easily!
- Environment includes: a true native Power Mac implementation of the C++ language, MrC/MrC++ compilers for fast Power Mac executable code, Visual Architect for fast, easy GUI generation, a new THINK project management system that supports complex applications, a new editor/browser , a powerful, easy-to-use source code debugger.
- Also includes: The industry-standard THINK Class Library!
- Next two updates free when you send in your registration card and more!

List $499.00 Our Price **$349.00** (SSYMCPP)



Symantec C++ for 68k

• The standard in development languages. • Powerful combination of fully integrated visual tools and the latest in C and C++ compiler technology. • Provides an object-oriented approach to application development. • Write code that is extensible, reliable, and maintainable. Includes: Integrated Environment with full source-code debugging, Incremental Linker which eliminates long link times, THINK Class Library, AppleEvents, Visual Architect™, THINK Inspector, Project Models, Source-Code Control with integration with Apple's SourceServer (included), Support for Scripting, Powerful Standard Libraries which includes I/O Streams, ANSI standard C library, and sample programs. • Full source code is included – Create applications that run on 68K Macs and Power Macs (emulated) These applications can easily be migrated to native Power Mac, by trading up to Symantec C++ for Power Mac.

List $299.00 Our Price **$99.00** (SSYMCPP68K)

Think Pascal Version 4.0

by Symantec Corporation

- Great for professionals and students
- Fully integrated for rapid turnaround time – take advantage of System 7 capabilities
- Supports large projects, enhanced THINK Class Library, System 7 compatibility, superior code generation, and smart linking.
- Includes four Macintosh disks, a user manual, and an object-oriented programming manual.

Our Price **$169.00** (SPASCAL)

LS Object Pascal CD-ROM by Fortner Research.

- Includes the world's first Object Pascal compiler for Power Macintosh
- 100% compatible with Apple's MPW Pascal
- Combines the best of Apple's native development tools with innovative new technology developed at Language Systems
- Compiler options specify 68K or native PowerPC code generation
- CD includes: LS Object Pascal compiler, Universal Pascal Toolbox interfaces, fully loaded MPW 3.3.1, 68K and PowerPC source debuggers, PowerPC assembler, online documentation, Macintosh Tech Notes, and a special version of AppMaker by Bowers Development that generates native Pascal source code.

List $399.00 Our Price **$349.95** (SOBJPASC)

MachTen – Power UNIX by Tenon Intersystems

- Dynamically linked shared libraries, memory mapped file access and integrated UNIX and Macintosh development tools.
- BSD 4.4 and conforms to the Federal Information Processing Standard 151-2 (the POSIX FIPS).
- Pre-emptive multitasking for UNIX applications and includes a full featured high-performance TCP/IP protocol stack that supports multi-homing and multi-casting, features not yet available even with Apple's new Open Transport.
- A complete UNIX software development environment with a source-level debugger and C, C++, and Fortran compilers all generating native PPC code.
- Also included is a high-performance X server and complete X11R5 X.

Our Price **$695.00** (SM10PPC)

SEE RELATED CATEGORY: Internet Related

Personal MachTen for 68K Macs

- MachTen UNIX for Macintosh (from Classic on up, including PowerBooks and Duos)
- A Mach-based Berkeley UNIX with pre-emptive multi-tasking
- Extends the Mac OS with UNIX networking and software development tools.
- Built-in internet services include domain name service, POP mail service, internet routing, SLIP & PPP, and Web service.

Our Price **$495.00** (SM10PER) Also Available: MachTen XWindows Our Price **$350.00**

SEE RELATED CATEGORY: Internet Related

Mjølner BETA System

• Software development environment supporting object-oriented programming in the BETA programming language. • Uniquely expressive and orthogonal. • Unifies just about every abstraction mechanism – including class, procedure, function, coroutine, process and exception. • Includes: general block structure, strong typing, whole/part objects. The compiler: binary code generation, automatic garbage collection, separate compilation, interface to C, Pascal, and assembler. • The system: persistent objects, basic libraries with containers classes, platform-independent GUI application frameworks on Unix, Mac and Windows NT, metaprogramming system. • The Mjølner BETA System for Macintosh requires MPW (basic set) 3.2 or later. Package containing compiler, basic libraries, persistent store, GUI framework, and comprehensive documentation. (Other packages are also available).

List $295.00 Our Price **$245.00** (SMJOLNER)

LS Fortran by Fortner Research

• 68K or PowerMac LS FORTRAN comes in two versions: one that generates 68K code (with or without FPU) and one that generates native code for the PowerMac • Source code is compatible between these two compilers • Programs originally developed with the 68K version can boost execution speed by 4 to 6 times by running the same program on a PowerMac • Supports all VAX intrinsic functions and data types, and virtually all VAX extensions • Built-in Debugging • Free Runtime Licenses • Full-featured editor, multiple window capabilities, and a powerful scripting language • Highly optimized, full ANSI standard, 32-bit clean, FORTRAN 77 compiler • Fully compatible with virtual memory under System 7 • Supports a variety of third-party tools to customize any FORTRAN environment • Power Tools include graphing software, subroutine libraries, and user interface tools • Free technical support.

Our Price **$695.00** (SLSFORT)

Fortran 77SDK by Absoft

For Power Macintosh includes a globally optimizing native compiler and linker, native Fx™ multi-language debugger, and Apple's MPW development environment.

• The compiler is a full ANSI/ISO Fortran 77 implementation, and includes all MIL-STD 1753 extensions, Cray/Sun-style POINTER, and several Fortran 90 enhancements • MRWE, Absoft's framework library is included in the MIG graphics library • Supports the native Macintosh PPC toolbox • Includes Absoft's Fx debugger which can debug intermixed FORTRAN 77, C, C++, and PPC assembler • The linker compiler, and debugger all run as native PPC tools, and produce Macintosh PPC executables

Our Price **$699.00** (SF77)

NS BASIC for the Newton

- A fully interactive implementation of the BASIC programming language.
- Runs entirely on the Newton – no host is required.
- Create files, access the built-in soups, and the serial port for input and output.
- Work directly on the Newton, or through a connected Mac/PC and keyboard.

Our Price **$94.99** (SNSBASIC)

LPA MacProlog

- Comprises a Edinburgh syntax Prolog compiler system set in an attractive multi-window development environment with an integrated program editor, graphical call-graph facilities and an interactive source-level debugger.
- Features high-level access to the Macintosh ToolBox for using graphics, dialogs, windows, icons, resources in a simple and versatile way.
- Includes interfaces to C and Pascal code resources.
- Enables the production of double-clickable distributable applications.
- Optional add-ons tools include flex, Prolog++, MacDBI for Oracle and the MacProlog Dialog Editor.

Programmer Edition **$745.00** (SLPAP) Developer Edition (includes the run-time generator and distribution license) **$1500.00** (SLPAD)

MacFortran II V 3.3 by Absoft

MacFortran II is a VAX/VMS compatible full ANSI/ISO Fortran 77 compiler including all MIL-STD 1753 extensions

- Comes with the latest version of MPW, and includes SourceBug, and SoftwareFPU
- Includes Absoft's Macintosh runtime Window Environment (MRWE) application framework, and MIG graphics library
- MacFortran II features improved 68040CPU support and is fully compatible with Power Macintosh under emulation

List $595.00 Our Price **$549.00** (SFORT2)

CodeWarrior Software Development Using PowerPlant by Jan L. Harrington

• C++ programmers will learn to develop object-oriented software applications for the Mac and Power Mac using the PowerPlant environment and the classes that support it. • Covers CodeWarrior 8. • Included CD-ROM contains source code for all the programming examples in the book and Metrowerks CodeWarrior Lite.

List Price: $34.95 Our Price: **$31.45**

Order Toll-free
800-MACDEV-1
(800-622-3381)

Roaster Subscription™

From Natural Intelligence Inc.

Get the most out of Sun's new Java™ Programming Language!

Includes CD-ROM!

SEE RELATED CATEGORY: **Dev. Environments**

- Write, test and run Java™ applets on the Macintosh in a full-featured Mac development environment
- Features include: project window that includes a finder-like view of packages, Macintosh native compiler, source code editor with powerful search features and intuitive use interface, runtime engine for quick and easy applet testing.
- Requirements : PowerPC, CD-ROM.
- Price includes the current DR 2 release and the next 2 non-developer releases.

List $299.00 Our Price **$199.00** (SROAST)

Roaster™

From Natural Intelligence Inc.

Includes CD-ROM!

SEE RELATED CATEGORY: **Dev. Environments**

- Price includes the current DR 2 release, and the first non-developer release

List $179.00 Our Price **$129.00** (SROAST1)

TCP/IP Scripting Addition™

- Award-winning AppleScript scripting addition
- Allows you to write scripts using MacTCP™ commands in AppleScript™.
- Send e-mail or files through a script, check if users are logged on (via Finger), automate FTP, Gopher, NetNews, Telnet, and LPR, verify links in HTML documents, and quickly write many other TCP/IP client-server programs.
- Works with AppleScript 1.0 or later and MacTCP 2.0.4 or later. Compatible with Open Transport™ 1.1.

Our Price **$49.00** (STCP)

SEE RELATED CATEGORY: **Scripting**

OOFILE HTML Writer

SEE RELATED CATEGORY: **Tools, Libs & Utilities**

The first OODBMS framework to offer a complete solution for application authors. Replaceable backend database, currently Faircom's c-tree Plus for cross-platform royalty-free power. PowerPlant and other frameworks integrated with edit fields, database browsers and more. AppMaker users, generate complete applications. User-friendly syntax makes it easier to migrate from FoxPro and the non-OO world. Demo's on CodeWarrior and AppMaker CD's. **$895.00** (SOOFCTREE) for a once-off c-tree bundle, or **$1,095.00** (SOOFSUB) 1year subscription. HTML and character-mode report-writer **$195.00** (SOOFHTML) Full GUI report-writer, including HTML, is **$495.00** (SOOFGUI). Mac Platform Bundle - includes all Mac frameworks, c-tree, and Report-Writer. **$1,495.00** (SOOFBNDL).

Providing Internet Services via the Mac OS

by Carl Steadman and Jason Snell

Includes CD-ROM!

- Shows you how to provide Web pages, FTP, Gopher, e-mail, and more with the Mac OS.
- Includes CD-ROM containing all the Mac server software and utilities you need.

List: $34.95

Our Price: **$31.46**

Learn HTML on the Mac

by Dave Mark and David Lawrence.

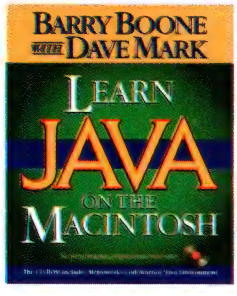
Includes CD-ROM!

- Shows you how to use HTML 3.0.
- Included CD-ROM contains Macintosh tools to design stunning multimedia Web pages.

List: $29.95 Our Price: **$26.95**

Learn Java on the Mac

by Barry Boone with Dave Mark.



- Easy-to-follow introduction for beginning programmers and Webmasters.
- Takes you through the core concepts of Java.
- Includes: Object-oriented techniques, unique features such as garbage collection, and basic programming concepts such as working with variables, threads and classes.
- Learn to apply this knowledge to write original Java applets for Web pages.
- Includes CD-ROM

List: $34.95 Our Price: **$31.45**

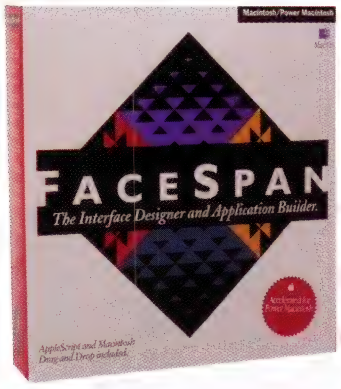
Includes CD-ROM!

For information: • Voice: 805/494-9797 • Fax: 805/494-9798 • E-mail: marketing@devdepot.com

DataScript

• Make your integrated AppleScript solutions database aware-today. • Takes just six lines of AppleScript to make new or existing scripted solutions database aware, and fetch data from RDBM's such as Oracle, Sybase, DB2, or Informix. • Easy to learn, easy to use, and easy to remember. "Inside DataScript" contains lots of easy to follow scripts to reuse in your own solution. • Very attractive licensing schemes.

List $249.00 Our Price **$229.99** (SWDSCRIPT)



FaceSpan™ v2

Build applications quickly and easily!

- Extensible Rapid Application Designer (RAD).
- Combines an interactive, visual interface design environment with the object-oriented power of AppleScript or any OSA language.
- Integrate the capability of scriptable programs into your custom application.
- Display scrolling lists, popup menus, scrolling text, movies, multi-column tables, pictures, icons, buttons, and others.
- Program custom objects using Pascal or C.
- Includes a royalty-free distribution license, for unlimited runtime users, of your FaceSpan-based applications. Plus a FREE UPGRADE to the next version for registered users!

List $199.00 Our Price **$179.00** (SFACESPAN)

Script Debugger by Late Night Software Ltd.

- A powerful and flexible AppleScript authoring tool – get the most from AppleScript!
- Advanced debugging environment offers single-step script execution with breakpoints.
- Script Debugger dictionary browser features a graphical view of objects provided by scriptable applications.
- Includes Late Night Software Scripting Additions – a collection of more than 70 new AppleScript commands, and Scheduler, a utility that allows you to launch scripts at pre-determined times.

List $129.00 Our Price **$119.00** (SDEBUG)

PreFab Player by PreFab Software, Inc.

- Lets your scripts query and control otherwise non-scriptable applications, desk accessories and control panels.
- Adds verbs that directly manipulate the Macintosh user interface: choose from menus & pop-ups, select radio buttons, type text, determine the name of the frontmost window, the state of a check box, etc. Balloon help identifies non-standard dialog items.
- Offers a simple, inexpensive way to distribute scripts and updates.
- Use a standard scripting system for automated testing.
- Requires: AppleScript or Frontier

Our Price **$95.00** (SPLAYER)

ScriptWizard™ 1.5

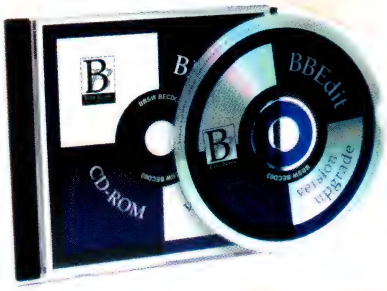
• Combines the power of a professional development environment with the Mac's ease of use. • Compatible with all Apple® Open Scripting Architecture languages, including AppleScript™. • Includes delivering testing and debugging facilities to improve your productivity. • Emphasizes features that speed script development. • Single-step scripts with true statement-level stepping, watch variable values as scripts execute, jump instantly to frequently used places in a script and find and replace specific text.

List $89.00 Our Price **$84.95** (SWIZ)

Scripter® by Main Event

- Powerful and easy to use, for both novices and experts.
- Includes Superior vocabulary access – point-and-click assembly of commands and object specifications; command window for experimentation.
- Includes Shortcuts and extended editing capabilities – extensive drag-and-drop, six-function find-and-replace; navigation markers; script library collection facility; many other timesavers for faster scripting.
- Interactive debugging – comprehensive variable watcher, expression evaluation, enhanced trace log, and real single step debugging!

List $199.00 Our Price **$179.00** (SSCRIPTER)



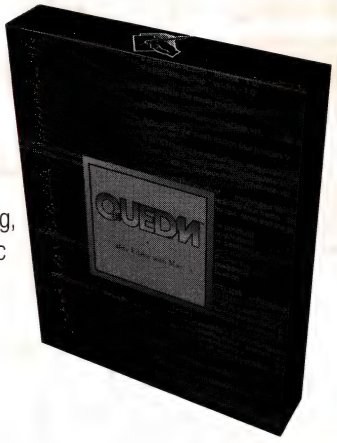
SEE RELATED CATEGORY:
Internet Related

BBEdit 4.0 from Bare Bones Software

- A powerful, easy-to-learn text editor.
- Adds new features for HTML coders, including a spelling checker and HTML tag palette
- Accelerated for Power Macintosh; dragging supported everywhere; Internet Config aware; PowerTalk aware.
- Integrated support for Symantec's IDE, Metrowerks CodeWarrior, THINK Reference 2.x, MPW ToolServer, and most other environments.
- Many UNIX style tools, including "grep" searches, file comparisons, and sorting. Multi-file search and replace.
- PopUpFuncs feature lets you jump to a function from a menu.

List $119.00 Our price **$94.00** (SBBEDIT)

QUED/M 3.0 by Nisus Software



- The programmer's text editor that defined the industry standard for speed and efficiency.
- Power PC native.
- Features integrated support for Symantec C/C++, Metrowerks CodeWarrior 6, and MPW.
- Supports all the major development environments on the Macintosh.
- Dozens of powerful editing features, including unlimited undo and redo, macro language, scripting, text folding, ten editable/appendable clipboards, markers, displaying text as ASCII codes, dynamic coloring of C/C++ keywords/comments, rectangular and non-contiguous selection.
- Includes Celestin Company's APPRENTICE 4.

List $119.00 Our Price **$69.00** (SQUEDM)

CMaster 2.0 by Jersey Scientific

- Installs into THINK C 5 / 6 / 7 and Symantec C++ for Macintosh and enhances the editor.
- Use its function popup to select a function and CMaster takes you right to it!
- Multiple clipboards and markers, a Function Prototyper, and a GoBack Menu which can take you back to previous editing contexts.
- Almost all features bindable to the keyboard, along over a hundred keyboard-only features like "Add New Automatic Variable." Glossaries, AppleScript and ToolServer support, Macros, and External Tools you create too!

Our Price **$129.95** (SCMASTER)

SEE RELATED PRODUCTS:
Quick Time Starter Kit

Movie Cleaner Pro 1.2 by Terran Interactive

- Compress QuickTime movies
- Powerful and easy-to-use
- Includes: drag and drop batch processing, suspend and resume, adaptive noise reduction, IMA audio compression, high quality crop and resize, A/V fades, gamma correction, de-interlacing and more!
- Automatically suggests the best settings for your application.
- Great for both beginners and experts
- Essential for CD-ROMs or Web sites

Our Price **$189.95** (SMOVIE)

SEE RELATED PRODUCTS:
Quick Time Official Guide

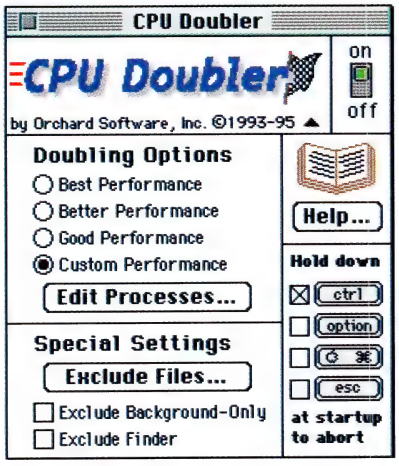
One & 2-Day SHIPPING
(call for details)

CodeManager by Metrowerks

SEE RELATED CATEGORY:
Dev. Environments

- Source code control system for the Macintosh.
- Based on and compatible with Microsoft Visual SourceSafe version 4.0.
- Works with any type of binary file including graphic, database, library and executable files.
- Includes support for multi-platform projects, security features, reverse delta versioning of files, comment areas for each revision, project analysis and reporting tools.
- Sold on a subscription basis with two future product updates.
- **Includes a 1 year MacTech subscription.**

Our Price **$399.00** (SCDMGR)



CPU Doubler by Orchard Software

SEE RELATED PRODUCTS:
Cron Manager

- Performance enhancement utility for the Macintosh.
- Increases the speed of your computer by 100%.
- Works on both the PowerPC and 68K Macintosh.
- Manages computer throughput using a proprietary scheduling algorithm.
- Ensure optimal performance and compatibility.

Our Price **$79.95** (SCPU2X)

CLImate by Orchard Software

- Command line interface that lets you communicate with your Macintosh using English commands to create, delete, rename, and move files and folders.
- Start applications, format disks, restart your computer, etc.
- Supplements the Finder.
- Includes a BASIC interpreter that lets you script your Macintosh without AppleScript.
- Includes advanced programming constructs: repeat loops, if/then/else conditionals, subroutine calls, etc.
- Implements wildcard characters, enabling you to work on groups of files.
- Comes bundled with sample programs and full documentation.
- Includes a free copy of Cron Manager – chronological event management utility.

Our price **$59.95** (SCLIMATE)

FILE genie PRO

File Genie Pro by Duet Development

- File Genie Pro lets you search ALL your developer CD-ROMs at once!
- Every Dev CD, OS SDK, CodeWarrior CD, E.T.O., and Bookmark CD you insert is cataloged automatically. A typical Dev CD with 10,000 files is cataloged in under 10 seconds.
- You can also catalog opticals, SyQuests, and floppies.
- File Genie Pro quickly finds any file on any disk by searching hard disks, network servers, and catalogs. Open, show, print, and more. View text and graphics files without opening other applications.
- When acting on an ejected disk file, File Genie Pro tells you which disk to insert, then continues automatically.

Our Price **$89.00** (SGENIE)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Guide Composer™ 1.2 by StepUp Software NEW!

SEE RELATED PRODUCTS: Real World AppleGuide
SEE RELATED PRODUCT: AppleGuide Complete
SEE RELATED PRODUCT: Danny Goodman's AppleGuide Starter Kit

- Create powerful Apple Guide help systems for any new or existing Macintosh application.
- Great for commercial developers, shareware developers, in-house developers, and consultants.
- Provides a WYSIWYG development environment: Guide content is developed in Guide windows.
- Design topics, phrases, and panels in the same format as the user will use them. Features are WYSIWYG interface, Topics, phrases, and hierarchical phrases, Coach marks, Fully-Integrated with Apple's Guide Maker (distributed with Guide Composer), compiles scripts automatically, PICTs in Panels, Generated Guide scripts are modifiable.
- Compiled files are 100% AppleGuide-compatible and royalty-free. Easy-to-use.

STEPUP SOFTWARE

New features include:

- Add up to six radio buttons on an Apple Guide panel, each of which can branch to another sequence or topic
- Add up to six checkboxes on an Apple Guide panel, each of which can insert another sequence or topic into the current topic and much more!
- FREE Update to all registered Guide Composer users. Demo is available at http://www.guideworks.com/.

Our Price **$99.00** (SGCOMP)

ScriptGen Pro by StepUp Software

- Installer script generator which requires no programming or knowledge of Rez.
- Supports StepUp's InstallerPack, StuffIt decompression, Compact Pro decompression, custom packages, splash screens, network installs, and resource installation.

Our price **$169.00** (SSCRPTGEN)

Step-Up Installer Pack by StepUp Software

- Package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts.
- Compression formats supported are Compact Pro & Diamond.
- Each atom also available separately.
- Compression requires additional licensing.

Our price **$219.00** (SINSTALL)

CronManager by Orchard Software

SEE RELATED PRODUCTS: CPU Doubler

- Implements the UNIX Cron facility.
- Open any Macintosh file on a given date and time.
- Simple interface.
- Works with any Macintosh file.
- Cron Manager bundled with CLImate,

Our price **$26.95** (SCRONMGR)

Tenon Ported Application CD

- Contains hundreds of applications that have been pre-compiled to run on Personal or Professional MachTen 68k systems, such as Unix, Perl, and TCL/TK
- Includes many internet servers such as Gopher, WAIS Ported Application CD Vol. 1 (for 68K Macs).

Our price **$49.95** (SPORTED)

Rosanne

SEE RELATED CATEGORY: Scripting

- A collection of utilities which offer the user complete control over raw data.
- Sort files, extract selected records, summarize frequency counts, create sample files, perform matching on multiple files, and reformat data to new specifications, all on the desktop, and even on files of a million records or more.
- Supports AppleScript™, enabling the user to link several actions together to complete an entire process.
- Recordable – users may perform a series of actions, and see them translated directly into AppleScript commands.
- Supports multi-tasking and background processing.
- Rosanne Utilities: Copy, Format, Select, Recode, Sort, Match, and Aggregate

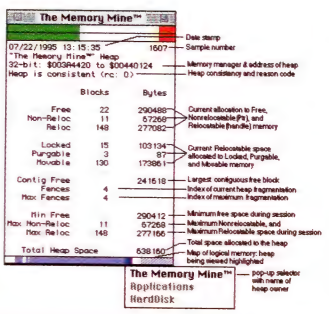
Our Price **$595.00** (SROSANNE)

Memory Mine by Adianta

SEE RELATED CATEGORY: Dev. Environments

- Monitor heaps, identify problems such as memory leaks, and stress test applications
- Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more
- Allocate, Purge, Compact, and Zap memory lets users stress test all or part of a program
- Works on Macintoshes with 68020 or later and System 7.0 or later.

List $99.00 Our Price **$94.99** (SMEMMINE)



QC by Onyx Technology. High performance runtime stress testing for applications.

- Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking.
- Extremely user friendly – ideal for non-programmer testers.
- Also available in Japanese.

List $99.00 Our price **$94.99** (SQC)

SEE RELATED CATEGORY: Dev. Environments

Last Resort Programmers Edition

- Records every keystroke, command key and mouse event (in local coordinates) to a file on your hard disk
- Great for program testing & debugging, and for technical support and help desks
- Last Resort keystroke files and recovers what you typed – great for data losses caused by power failure, crashes, or accidental deleting.

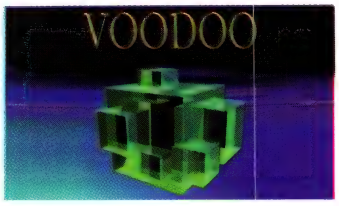
Our price **$74.95** (BLSTRSRT)

Spyer by InCider

- Easy to use tool that records all actions (including mouse movement) you perform on a Macintosh computer and then replays them at your preferred speed.
- Recorded data can be saved in files for future use.
- Works as a background process with any Macintosh application and is triggered by user defined Hot Keys.
- Enables the "Continuous Redo" utility and is especially useful for software testing and demonstration.

Our price **$39.00** (SSPY)

VOODOO



- Version control tool for the simple and clear management of projects in which files are created in numerous versions (variants and revisions).
- Allows both variant and revision control, and it manages not only variants and revisions of single files, but of a whole software project (multi files, multi users, multi variants, access rights, etc.)
- Graphical user interface and is not only suitable for mere source code control but can handle all different kinds of files with amazing compression rates: typical size of delta between arbitrary files 5%
- Please note special prices for multiple copies:

Single license **$229.00** (SVOODO01); 2 pack **$359.00** (SVOODO02); 5 pack **$799.00** (SVOODO05); 10 pack **$1369.00** (SVOODO10); 20 pack **$2399.00** (SVOODO20)

Additional pricing available on request.

SoftPolish CD-ROM by Bare Bones Software

- The essential tool for software quality assurance on the Macintosh
- Helps you identify inconsistencies with Apple's user interface guidelines, misspelled words, missing resources, and other mistakes
- Provides tools to put the finishing touches on software distribution packages prior to release
- Works independently of any programming language or environment
- Ideal for sanity checking software throughout the development process.

List price: $99 Our price: **$89** (SSOFTPOL)

LJ Profiler by Lars Jordebo Datakonsult

- Supports profiling of C++ 68K and PowerPC applications compiled with CodeWarrior, CFront or Symantec C++.
- Based on active profiling, i.e. profiling code called at function enter and exit, the browser application lets you follow call chain timings in hierarchical views or separate windows.
- Collect, organize, compare and save profiling data from different versions of your application into a project. Scriptable and recordable with full access to most internal data structures.
- Optional remote profiling and tracking of segment and stack usage. Full source code to what you link into your application.

Our price **$295.00** (SLJPROF)

PowerTap™

- Accelerates software by tapping into multiple processors.
- Easy API – submit tasks and retrieve results!
- Full error recovery system plus scheduling algorithms for optimal assignments and fastest possible execution.
- Compatible with all Macintosh hardware, software and major compilers.

PowerTap/2, 2 remotes edition List $1,200.00 Our Price **$1,095.00** (SPOWTAP2)

PowerTap/5, 5 remotes edition List $1,900.00 Our Price **$1,795.00** (SPOWTAP5)

PowerTap/n, unlimited edition List $2,700.00 Our Price **$2,495.00** (SPOWTAPN)

One & 2-Day SHIPPING (call for details)

StoneTable

- Replaces all functions found in list manager
- Variable size columns/rows; different font, size, style, forecolor, backcolor per cell; sort, resize, move, copy, hide columns/rows; edit cells/titles in place; titles for columns/rows; multiple lines per cell; grid line pattern/color; greater than 32k data per table; up to 32k text per cell; support for balloon help and binary cell data. Versions for Think C, Think Pascal, MPW C, MPW Pascal, CodeWarrior 6 C.

StoneTable Extra

- Drag selected cells within table or to other tables.
- Add rows as part of drag; popup menus or check boxes in cells; variable width grid lines; move/drag/resize table in window; clipboard operations on multiple cells.
- Requires StoneTable.

This product can now be ordered in 3 different packages.

68K StoneTable

This includes StoneTable, StoneTableExtra for Think Pascal, Think C, MPW C, MPW Pascal, CodeWarrior C, CodeWarrior Pascal

Our price **$175** (SSTABLEX)

PPC StoneTable

This includes StoneTable PPC, StoneTableExtra PPC for Think C, MPW C, MPW Pascal, CodeWarrior C, CodeWarrior Pascal

Our price **$175** (SSTNXPPC)

68K/PPC StoneTable

Includes both packages

Our price **$325** (SSTONEFAT)

PictureCDEF 1.3 by Paradigm Software

- Professional-level CDEF for creating custom graphical buttons (8-64 pixels) – used in products by Adobe, ProVue, STF Technologies and others!
- Multi-monitor and bit-depth sensitive.
- The button graphic (cicn, ResEdit) can be changed at runtime and even animated with a call-back routine.
- Create distinct buttons in seven variations: MultiState, PushButton, FlexiButton, ToggleButton, ChkButton, PushPictButton and TogglePictButton.
- Manual, sample code and MacApp 3.0 support included.

Full source code: **$95.00** (SPCDEF)

One & 2-Day SHIPPING (call for details)

SpellsWell 7 1.0.4

- Award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and Fair Witness).
- Checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double word errors, abbreviation errors, a/an before vowel/consonant, etc.
- MacTech orders include developer kit with Writeswell Jr., a sample AppleEvents Word Services word-processor and its source code.

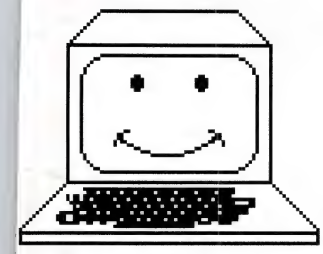
Our price **$74.95** (SSPELL)

MacWireFrame by Amplified Intelligence

- Create your own virtual reality application with MacWireFrame, a virtual reality application framework.
- Includes a complete library of object oriented graphics routines.
- Easy to understand application frame work, plus an example application program that lets you start solid modeling right away.
- Complete with fully documented source code.
- Guaranteed $49.99 upgrade to the soon to be released, scriptable, MacWireFrame 5.0.

List $299.00 Our Price **$75.00** (SFRAM)

Order Toll-free
800-MACDEV-1
(800-622-3381)



B-Tree HELPER™ 2.2

- Inexpensive database engine for Macintosh programmers in C source code.
- Uses contiguous fixed length blocks.
- Expands the file as necessary and contracts files when possible.
- Inserts and deletes keys in one or more B-Trees.
- Finds keys equal to, less than, or greater than a given value in a few hundredths of a second.
- Finds lists of records whose keys are equal to, less than, or greater than a given value or are in a range of values.

Our Price **$150.00** (SBTREE)

ViviStar CONSULTING

Q3S/3dPane/SmartPane

- Full featured 3d graphics library.
- Polyhedra; Gouraud shading; stereoscopic projections; pipeline access; animation and model interaction support; a "triad mouse" to map 2d mouse movement to 3d; and all the regular 3D features.
- 3dPane provides integration with the TCL and provides a view orientation controller. SmartPane provides TCL offscreen image buffering, flicker free animation, and QuickTime movie recording. SmartPane functions in 3d or 2d scenarios.
- All work with C++ compilers or ThinkC 6 and compile to PowerPC or 68K target machines.

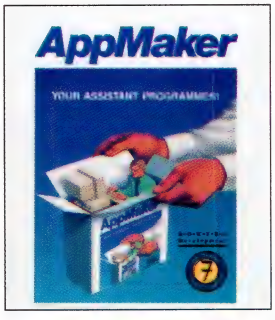
Our Price **$192.00** (SQ3)

dtF

- True relational database system for Apple Macintosh computers.
- Provides a powerful choice for developers who want to create database centered applications with no performance trade-offs.
- Features SQL, full transaction control, error recovery, single user, client server architecture and multi-platform support including DOS, Windows, OS/2 and UNIX.
- The C/C++ API is identical and fully portable across all supported platforms.
- Third-party vendors supporting dtF will be able to offer a variety of advanced features and benefits to their customers royalty free.
- Tools are included for importing, exporting, creating and managing databases and users.
- Supported development environments include: Symantec, MPW, Metrowerks and more. Mac/SDK

List $695.00 Our Price **$679.00** (SDTF)

AppMaker



- Develop the user interface for a Macintosh application using the original interface builder.
- Just point and click to design your application.
- Creates resources and generates excellent source code.
- Supports most development environments including Metrowerks, Symantec, or MPW;
  C, C++, or Pascal; procedural or object-oriented, using PowerPlant, TCL, or MacApp.
- The generated code uses the Universal Headers to provide PowerMac compatibility.
- Great tool for beginners to learn object-oriented and Macintosh Toolbox programming techniques.
- Includes one-year subscription on CD.

List $299.00 Our Price ~~$279.99~~

SPECIAL PRICE! **$149.00** (SAPPMAKE)

Includes CD-ROM!

NeoAccess™

- Full-featured object database engine for use in Macintosh, Windows, Unix and DOS based C++ applications.
- Extended binary trees and binary search algorithms tuned for short access times; dynamically combined, collapsed, and compressed indices; object caching for instant access to previously used objects.
- Build fast, powerful applications in record time!

Our price **$749.00** (SNEO)

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

TOOLS, LIBRARIES & UTILITIES

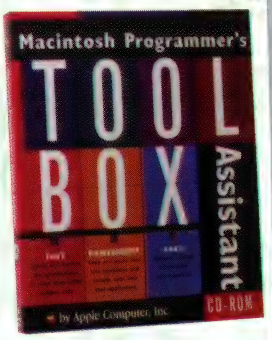
Developer DEPOT™

Complete info on these products and hundreds more! http://www.devdepot.com

Web site: http://www.devdepot.com • E-mail: orders@devdepot.com

Inside Macintosh®: Programmer's Toolbox Assistant CD-ROM

Instant electronic access to Inside Macintosh essentials.



- Get quick access to reference pages for over 4,000 Toolbox calls in your system software from their development environment.
- Essential information for Macintosh software developers.
- Hypertext links allow programmers to view related topics easily.
- The ultimate electronic reference tool for Macintosh programmers.

List $99.95 Our Price **$49.99** (STBASST)

Inside Macintosh®: Overview by Apple Computer, Inc.

- An overview of Macintosh programming fundamentals and a road map to the New Inside Macintosh library.
- Covers various programming tools and languages, compatibility guidelines and considerations for worldwide development. 176 pages.
- Great for beginners!

List $22.95 Our Price **$11.50** (BIMOVER)

Inside Macintosh®: Macintosh Toolbox Essentials

by Apple Computer, Inc.

- Covers the heart of the Macintosh. The toolbox enables programmers to create applications consistent with the Macintosh "look and feel".
- Describes Toolbox routines and shows how to implement essential user interface elements, such as menus, windows, scroll bars, icons and dialog boxes. 880 pages.

List $34.95 Our Price **$17.50** (BIMTBOX)

Inside Macintosh®: More Macintosh Toolbox by Apple Computer, Inc.

- Managers discussed include Help, List, Resource, Scrap and Sound.
- Covers other Macintosh features such as how to support copy and paste, provide Balloon Help, play and record sound and create control panels.

List $34.65 Our Price **$17.33** (BIMMAC)

Inside Macintosh®: Files by Apple Computer, Inc.

- Describes the parts of the operating system that allow you to manage files.
- Explains how your application can handle the commands typically found in a File menu.
- Provides a reference to the File and Alias Managers, the Disk Initialization and Standard File Packages. 510 pgs.

List $29.95 Our Price **$14.99** (BIMFIL)

Inside Macintosh®: Files by Apple Computer, Inc.

- Describes the parts of the operating system that allow you to manage files.
- Explains how your application can handle the commands typically found in a File menu.
- Provides a reference to the File and Alias Managers, the Disk Initialization and Standard File Packages. 510 pgs.

List $29.95 Our Price **$14.99** (BIMFIL)

Inside Macintosh®: Operating System Utilities by Apple Computer, Inc.

- Describes parts of the Macintosh Operating System that allow you to manage various low-level aspects of the operating system.
- Shows how to get information about the operating system, manage operating system queues, handle dates and times, control the settings of the parameter RAM, manipulate the trap dispatch table, and receive and respond to low-level system errors.

List $26.05 Our Price **$13.00** (BIMOPSU)

Inside Macintosh®: Processes by Apple Computer, Inc.

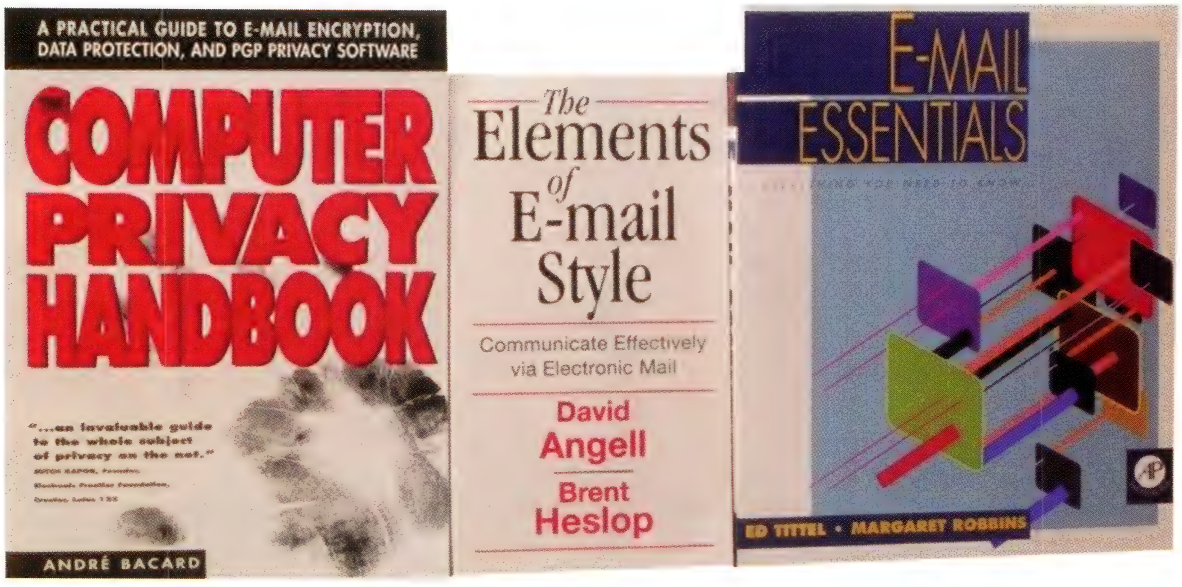
- Describes the parts of the Macintosh operating system that allow you to control the execution of processes and interrupt tasks.
- Shows how you can use the Process Manager to get information about processes loaded in memory.
- A reference for the Vertical Retrace, Time, Notification, Deferred Task, and Shutdown Managers. 208 pages.

List $22.95 Our Price **$11.50** (BIMPROC)

Inside Macintosh®: Memory by Apple Computer, Inc.

- Describes the parts of the Macintosh operating system that allow you to manage memory.
- Provides detailed strategies for allocating and releasing memory, avoiding low-memory situations, reference to the Memory Manager, the Virtual Memory Manager, and memory-related utilities. 296 pages.

List $24.95 Our Price **$12.50** (BIMMEM)



The Computer Privacy Handbook

- A practical guide to e-mail encryption, data protection, and PGP privacy software.

List $24.95 Our Price **$22.45** (BPRIV)

The Elements of E-Mail Style

by Brent Heslop and David Angell

- Write solid, effective E-Mail and avoid common pitfalls.

List $14.95 Our Price **$13.45** (BEMAIL)

E-Mail Essentials

by Ed Tittel & Margaret Robbins

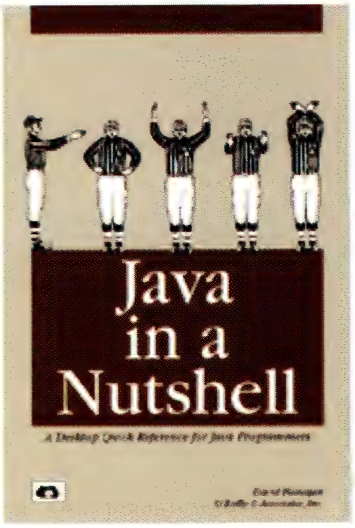
- A hands-on guide to the basics of e-mail.

List $24.95 Our Price **$22.45** (BEMAILE)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Cyberpunk Handbook,

List $9.95 Our Price **$8.95** (BCYBPUNK)



Java in a Nutshell

- A complete quick reference guide to Java, the hot new programming language from SunMicrosystems.
- Contains descriptions of all of the classes in the Java 1.0 API, with a definitive listing of all methods and variables.
- Also contains an accelerated introduction to Java for C and C++ programmers who want to learn the language fast.

List $14.95 Our price **$13.45** (BJAVANUT)

The Instant Internet Guide

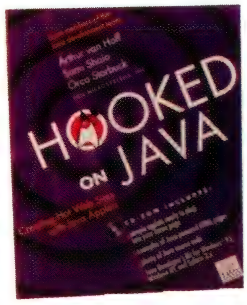
List $14.95 **$13.45** (BINSTANT)

Planning and Managing Websites

- The definitive guide to setting up and running a Web site on the Macintosh.
- Learn everything you need to know about using WebSTAR, the best known HTTP server software and its shareware predecessor MacHTTP.
- Write CGI applications for your server – in AppleScript and in C.
- CD includes a special version of WebSTAR, plus tons of useful software.

List $39.95 Our Price **35.96** (BPLANWEB)

SEE RELATED CATEGORY:
Internet Related



Hooked on Java

- Written by the Java development team at Sun.
- An introduction to using applets, for Web administrators, designers, and developers.
- Demonstrates how to use applets in your own pages.
- Includes a concise introduction to the Java language, and a CD with tools.

List $29.95 Our Price **$26.95** (BHJAVA)

Teach Yourself Java for Macintosh in 21 Days

- Add interactivity and multimedia to Web pages!
- Astep-by-step guide to make your Web site come alive.
- Learn the basics of programming Java applets and the concepts behind the Java language.
- Includes CD-ROM with a limited version of Roaster, the first commercial, integrated applet development environment for Java for the Macintosh!

List $40.00 Our price **$36.00** (BJAVAMAC)

JavaScript for the Macintosh

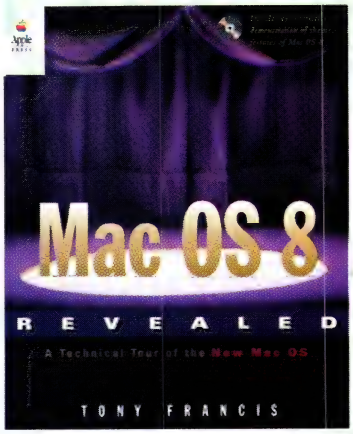
- Allows non-programmers to take advantage of the power of Netscape Navigator.
- Expand the capabilities of your Web page, without having to understand C or C++.
- CD-ROM contains "Wizlets" that allows you to easily create your own JavaScripts
- Takes you step-by-step through programming cross-platform JavaScripts
- Details how to create JavaScripts for JavaScript-aware Web browsers

List $45.00 Our price **$40.50** (BJAVASCRPTJ)

Mac OS 8 Revealed by Tony Francis

* The first authoritative look at this exciting new operating system.
* A must for Mac developers who want to make their software compatible with Mac OS 8.
* essential for system administrators who plan to upgrade their system.
* Included CD-ROM contains demos of new Mac OS 8 features.

List: $34.95 Our Price: **$31.45**

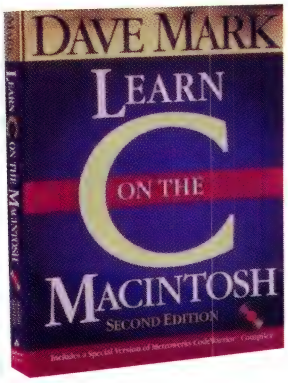


Designing AppleTalk Network Architectures

- Network Frontiers Field Manual Series by Dorian J. Cougias, Tom Dell & E.L. Heiberger

* Learn to update your current network design, or create a new one.
* Thoroughly explains AppleTalk, and provides a complete overview of cabling and hardware issues to maintain a succesful network.
* Includes CD-ROM

List Price: $29.95 Our Price: **$26.95**



Learn C on The Macintosh Second Edition By Dave Mark

- New revised edition.
- Easy-to-understand – everything you need to start programming!
- Updated and enhanced exercises that lead you step by step. You'll learn function, variables, pointers datatypes, data structures, file input and output and more!
- Includes CD-ROM with Metrowerks CodeWarrior™ Lite – the hottest Macintosh programming environment (including a PowerPC version).

List $34.95 Our Price **$31.45** (BLEARNC2)

SEE RELATED CATEGORY:
Dev. Environments

Discover Programming for Macintosh

- Includes full working version of CodeWarrior along with three online tutorial books and Dave Mark's "Learn C on the Macintosh" converted to AppleGuides.
- Includes C, C++ and Object Pascal compilers for generating 68K Macintosh code, source-level debuggers, object-oriented frameworks (PowerPlant, MacApp), Apple's MPW, complete online documentation and source code examples for all languages and platforms.
- The IDE software has been localized in eight languages plus English. This product is not sold as a subscription.
- **Includes a 3 month subscription to MacTech Magazine.**

Our Price **$79.00**

SEE RELATED CATEGORY:
Dev. Environments

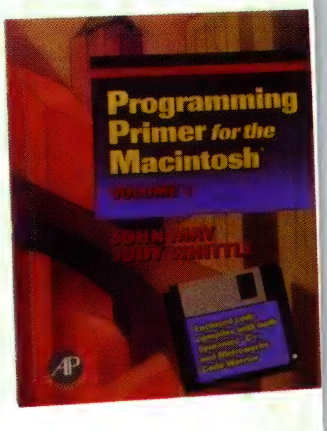
ScriptBase™ by Main Event Software

- A database for storing persistent objects to be made available for access to AppleScript, Apple's system-level user scripting language for controlling applications on Macintosh® computers.
- Once installed, the database becomes part of the AppleScript system, adding a host of commands to the basic AppleScript vocabulary. Retrieving the objects is simple using AppleScript's natural-language syntax and structure. Objects stored and retrieved in ScriptBase can be accessible any time from any script on the user's computer. These objects can be of any type, including numbers, character strings, lists, records, scripts, and references to disks, files, folders, as well as abstract raw data, to name just a few.
- ScriptBase can be used to maintain system-wide settings, such as sets of preferences, paths to frequently-used files or folders. Complex installations can be made easier by organizing data and scripts within the database's structure.

Our Price **$79.00** (SSCPTBASE)

Macintosh C Programming Primer Volume I

Second Edition, Inside the Toolbox Using THINK C by Dave Mark and Cartwright Reed



- Updated new edition of the Macintosh programming best seller.
- System 7, new versions of THINK C and ResEdit.
- Learn how to use the resources, Macintosh Toolbox and interface to create stand-alone applications.
- 672 pages.

List $26.95 Our Price **$24.25** (BCPRIM1)

Macintosh C Programming Primer Volume II

Mastering the Toolbox Using THINK C by Dave Mark.

- Covers advanced topics such as: Color QuickDraw, THINK Class Library, TextEdit, and the Memory Manager: 528 pgs.

List $26.95 Our Price **$24.25** (BCPRIM2)

Learn C++ on the Macintosh by Dave Mark.

SEE RELATED CATEGORY: Dev. Environments

- Basic syntax of C++ and object programming.
- Learn how to write, edit, and compile your first C++ programs.
- Features key C++ concepts such as derived classes, operator overloading, iostream functions and more.
- Includes a special version of Symantec C++ for Macintosh. Book/disk package with 3.5" 800K Macintosh disk. 400 pages.

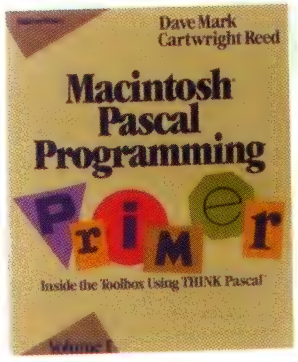
List $36.95 Our Price **$33.26** (BLRNCPP)

Macintosh Pascal Programming Primer Volume I

Inside the Toolbox Using THINK Pascal by Dave Mark and Cartwright Reed.

This tutorial shows programmers new to the Macintosh how to use the Toolbox, resources, and the Macintosh interface to create stand-alone applications with Symantec's THINK Pascal. 544 pages.

List $26.95 Our Price **$24.25** (BPASCPRI)



Power Macintosh Programming Starter Kit

by Tom Thompson.

SEE RELATED CATEGORY: Dev. Environments

- Enter the world of the PowerPC chips.
- Get the scoop on the microprocessors, the RISC architecture, and how to write native code and emulation operations to create software for the Macintosh PowerPC.
- CD-ROM includes a unique compiler for writing code easily.

Includes CD-ROM!

List $39.95 Our Price **$35.10** (BPPCSTART)

Macintosh Programming Secrets 2nd edition

By Scott Knaster, and Keith Rollin

- Macintosh Programming Secrets is divided in two parts.

Part 1, "Concepts and Ideas", discusses the evolution of the Macintosh and the standards, customs, and software that shape the system as well as the Macintosh user interface.

Part 2 "Technical Adventures", presents the skeleton of an application, and then builds upon that framework to describe how to: • Create fancy dialogue boxes • Utilize the new 32 bit QuickDraw developments • Track the mouse with "marching ants" • Manage multiple windows with the Window manager • Copy files within a program • Install the worlds strangest spinning cursor.

List $31.95 Our Price **$28.76** (BPSECRET)

Order Toll-free
800-MACDEV-1
(800-622-3381)

yberdog - The Complete Guide to Apple's Internet Productivity Technology

Jesse Feiler

Learn all you need about Cyberdog — Apple's new Internet productivity interface.
Turn everyday documents into glitzy masterpieces of content.
Provides guidance on extending and customizing Cyberdog by adding or replacing part editors.
Includes sample Cyberdog solutions.

List Price: $29.95 Our Price: **$26.95**

Cyberdog Programmers Kit by Apple Computers, inc.

- Apple's official reference.
- A must for developers who want to make customizable Internet access available to all Mac users.
- Included CD-ROM contains all the tools needed to create Cyberdog-aware components.

List: $34.50 Our Price: **$31.05**

Programming with AppleTalk by Michael Pierce

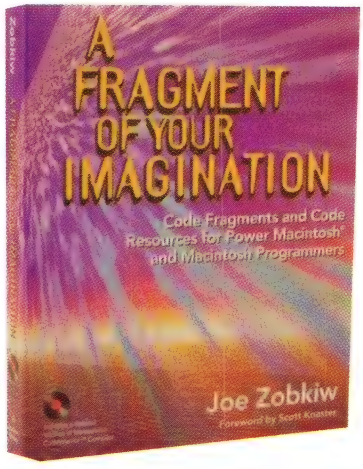
- Programming with AppleTalk is the hands-on guide to understanding and working with AppleTalk. Topics covered include:
- How to create applications and system extensions that run with AppleTalk.
- AppleTalk protocols and the protocol stack, transport media, the Preferred AppleTalk Intefrace, and the storage management.
- Numerous working code examples walk you through using RDEV, INIT, NBP, ATP, and ADSP You will also learn the use of: Synchronous, and asynchronous calls, How to avoid heap fragmentation, And how to configure a Chooser Interface.

List $24.95 Our Price **$22.45** (BPROAT)

A Fragment of Your Imagination by Joe Zobkiw

- Packed with useful code fragments for the Macintosh and Power Macintosh.
- Hard to find information about techniques used to structure and build fat, safe fat, and accelerated code resources.
- All code is reusable and is provided on the disc, along with Metrowerks Code Warrior Lite. Book/CD-ROM, 528 pages.

List $39.95 Our Price **$35.96** (BFRAG)

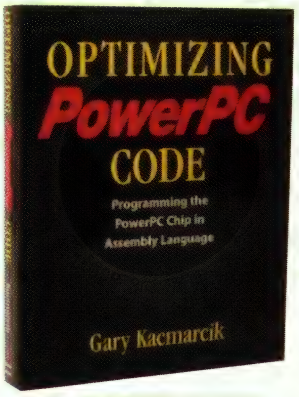


Programming QuickDraw

- Learn to build color pictures on the Mac
- Learn to maintain the highest degree of compatibility for you applications across the Mac platform
- Learn to perform sophisticated image processing operations with CopyBits( ) and CopyDesk( )
- Learn to enable your applications to take advantage of QuickDraw compliant graphics hardware and accelerators.

List $26.95 Our price **$24.25** (BPROQDRAW)

Optimizing PowerPC Code: Programming the PowerPC in Assembly Language



- Take full advantage of the potential of the PowerPC by mastering the Assembly Language techniques.
- Learn to produce faster more robust software!

List $39.95 Our Price **$35.96** (BOPTPPC)

Inside Macintosh®: PowerPC Numerics

by Apple Computer, Inc.

- Describes the floating-point numerics environment provided with the first release of PowerPC processor-based Macintosh computers.
- Provides a description of the IEEE standard 754 for binary floating-point arithmetic., and how RISC Numerics compiles with it.
- Shows programmers how to create floating-point values and how to perform operations on floating-point values in high-level languages such as C and in PowerPC assembly language.

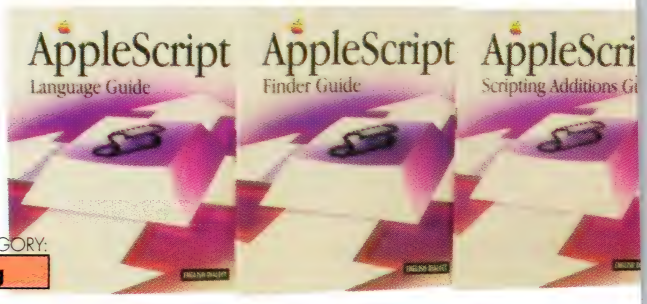
List $28.95 Our Price **$26.00** (BIMPPCNUM)

Inside Macintosh®: PowerPC System Software by Apple Computer, Inc.

- Describes the new process execution environment and system software services provided with the first version of the system software for Macintosh on PowerPC computers.
- Contains information to write applications that can run on the PowerPC.
- Shows how to make your software compatible with the new run-time environment provided on PowerPC-based Macintosh computers. It also provides a complete technical reference for the Mixed Mode Manager, the Code Fragment Manager, and the Exception Manager.

List $24.95 Our Price **$22.45** (BIMPPCSYS)

AppleScript Finder Guide, English Dialect by Apple Computer, Inc.



- Provides definitions for Finder object classes and commands.
- Write, record, or run scripts that trigger the same desktop actions that you trigger using the keyboard and mouse.

List $19.95 Our Price **$17.95** (BAFG)

SEE RELATED CATEGORY: **Scripting**

AppleScript Language Guide,

by Apple Computer, Inc.

- A complete reference for anyone using AppleScript to modify existing scripts or to write new ones.
- Contains useful information for programmers who are working on scriptable applications or complex scripts.
- Features detailed definitions of AppleScript terminology and syntax in the following categories: Value classes, commands, objects and references to objects, expressions, control statements, handlers, and script objects.
- Includes many sample scripts, discusses advanced topics such as writing command handlers for script applications, the scope of script variables and properties declared at different levels in a script, and inheritance and delegation among script objects.

List $29.95 Our Price **$26.95** (BALG)

SEE RELATED CATEGORY: **Scripting**

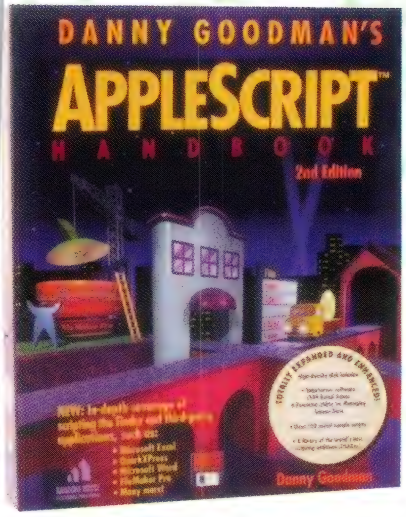
AppleScript Scripting Additions Guide

by Apple Computer, Inc.

- Use the standard scripting additions commands.
- Write scripting additions.

List $18.95 Our Price **$17.05** (BSCRADD)

SEE RELATED CATEGORY: **Scripting**



Danny Goodman's AppleScript Handbook Second Edition by Danny Goodman

- Customize and extend the capabilities of any Macintosh computer – no programming experience needed!
- Learn to use scripts to enhance the Macintosh environment, automate many processes, link data between applications, and much more.
- All-new examples showing how to integrate AppleScript with the Finder, spreadsheets, desktop publishing programs, graphics applications, databases, telecommunications programs, utilities, and HyperCard.
- Includes 3 1/2" disk with over $100 worth of software, including AppleScript 1.1, valuable utilities, and powerful, ready-to-use scripts.

Includes Disk!

List $39.00 Our Price **$35.00** (BDGASHB)

SEE RELATED CATEGORY: **Scripting** SEE RELATED CATEGORY: **Tools, Libs & Utilities**

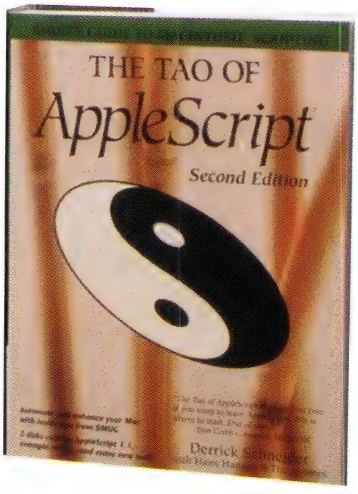
Applied Mac Scripting

SEE RELATED CATEGORY: **Scripting**

- Learn to design and develop powerful scripts.
- Covers AppleScript™, Frontier, QuicKeys, Tempo II, nShell, FaceSpan Application Builder, Scripting PlainTalk and System 7.5.
- Hands on tutorial shows you how to automate your Macintosh activities by learning how to use the AppleScript and Frontier scripting environments.
- Harness the capabilities of a wide variety of Macintosh applications into the integrated productivity tools. This includes such things as the newspaper script which combines the power of SITcomm, MacWrite Pro, and FileMaker Pro, or QuarkXPress.

List $34.95 Our Price **$31.45** (BAPPLIED)

The Tao of AppleScript: BMUG's Guide to Macintosh Scripting, Second Edition by Derrick Schneider & Hans Hansen



- A complete, natural introduction to AppleScript programming essentials.
- Learn to customize applications, automate tedious tasks, and create programs without having to use a complex programming language.
- Includes 2 disks containing AppleScript, QuickTime, StuffIt Lite, ResMover, and other helpful utilities.
- Loaded with practical examples for easy learning

Includes Disk!

List $29.95 Our Price **$26.95** (BTAO)

SEE RELATED CATEGORY: **Scripting**

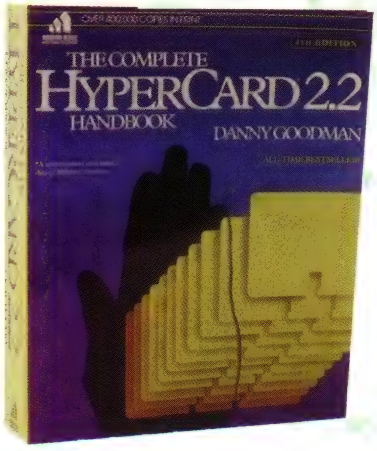
The Complete AppleScript Handbook by Danny Goodman

Master AppleScript with the definitive book/disk toolkit. This self-contained toolkit teaches you AppleScript from the ground up providing you with tools you need to automate tasks and integrate Macintosh applications. This book explains in detail all commands and usage of the AppleScript language including:

• Issuing command • Describing objects • Working with values, variables, and expressions • Using if-then constructions, loops and subroutines • Error checking and debugging • Scripting Finder-level processes • Using AppleScript with third party applications

List $35.00 Our Price **$31.50** (BAPLSCRHB)

SEE RELATED CATEGORY: **Scripting**



The Complete HyperCard® 2.2 Handbook Fourth Edition

by Danny Goodman

- The biggest-selling programming Mac book.
- Learn to build working applications using the latest version of HyperCard.
- Covers text, painting tools, extension commands (XCMDs), scripting in HyperTalk, and more.

List $35.00 Our Price **$31.50** (BHYPCRD2)

SEE RELATED CATEGORY: **Scripting**

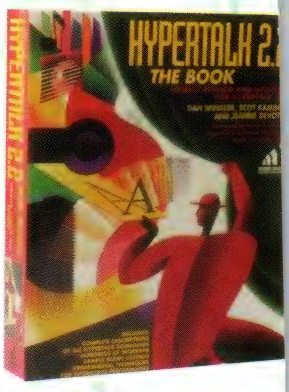
HyperTalk® 2.2: The Book Second Edition

by Dan Winkler, Scott Kamins, and Jeanne DeVoto

SEE RELATED CATEGORY: **Scripting**

- The most complete, authoritative source on HyperTalk 2.2 programming and troubleshooting
- Covers each language element of HyperTalk 2.2 (including the odd quirk or bug).

List $35.00 Ours Price **$31.50** (BHYPTAL)



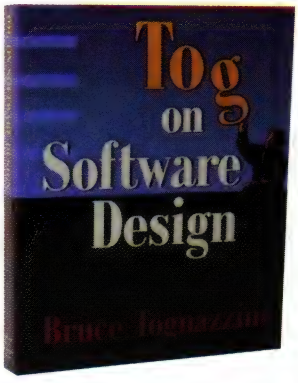
HyperCard Stack Design by Apple Computer, Inc.

HyperCard Stack Design Guidelines is an essential book for everyone who creates Apple® HyperCard stacks. Included are illustrated discussions of:

- Guidelines for stack development-audience evaluation, subject matter requirements and constraints, mode of presentation, and testing
- Navigation, the importance of making sure users can get around in your stacks
- Graphic Design and illustration- including the use of grids to determine card and background layout
- Text in stacks- placement, readability, and special considerations when writing for the screen
- Music and sound in stacks-as subject matter, reinforcement, entertainment, alert mechanism, or transition

List $21.95 Our Price **$19.95** (BHYPSTA)

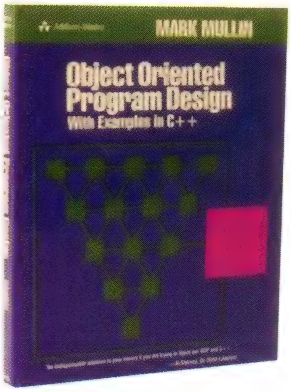
SEE RELATED CATEGORY: **Scripting**



Tog on Software Design by Bruce "Tog" Tognazzini

Respected industry futurist, Tog, presents his vision of our technological future, detailing the steps computer professionals need to take to deliver new technologies that will profit the industry and benefit society in general. Contains Tog's insights on a wide range of topics from quality management to the meaning of standards, and responses to queries supplied by designers and developers.

List $29.95 Our Price **$26.95** (BTOG)



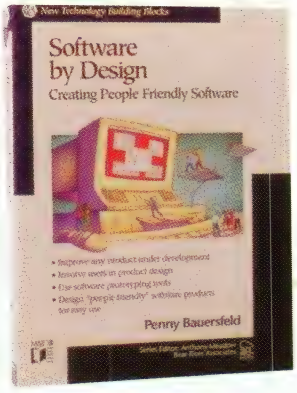
Object Oriented Program Design by Mark Mullin

- A concise guide to the essential concepts and techniques of OOP design
- Clarifies the key concepts of object oriented programming such as objects, classes, entities, hierarchies, and inheritance
- Uses typical database application to illustrate each OOP topic, to give the programmer a familiar point of reference

List $22.95 Our Price **$20.66** (BOOPRODES)

Software By Design: Creating User Friendly Software

by Penny Bauersfeld (Series Editor: Tony Meadow)



- A thorough how-to for designing software that is easy to learn and comfortable to operate.
- Written from the Macintosh perspective, but compatible with all platforms.
- Stresses user input from initial design, through prototyping, testing and revision.
- Provides tools for analyzing user needs and test responses, plus exercises for sharpening user-oriented design skills.

List $29.95 Our Price **$26.95** (BDESIGN)

Inside Macintosh®: CD-ROM by Apple Computer, Inc.

- More than 25 volumes in electronic form.
- Includes: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components.
- Access over 16,000 pages of information with Hypertext linking and extensive cross referencing.

List $99.95 Our Price **$49.99** (BIMCD)

Inside Macintosh®: Sound by Apple Computer, Inc.

- Describes the parts of the Macintosh system software that allow you to manage sounds.
- Contains information to write applications that can record and play back sounds, compress and expand audio data, convert text to speech, and perform other similar operations.

List $26.95 Our Price **$13.50** (BIMSOUND)

Inside Macintosh®: Imaging by Apple Computer, Inc.

- Covers QuickDraw and Color QuickDraw.
- Includes general discussions of drawing and working with color.
- Describes the structures that hold images and image information, and the routines that manipulate them.
- Covers the Palette, Color, and Printing Managers, and the Color Picker, Color Matching, and Picture Utilities.

List$26.95 OurPrice **$13.50** (BIMIMAG)

Inside Macintosh®: QuickTime by Apple Computer, Inc.

- For developers who want to create applications that use QuickTime, the system software that allows the integration of video, animation, and sounds into applications.
- Describes all of the QuickTime Toolbox utilities.
- Provides the information you need to compress and decompress images and image sequences.

List $29.95 Our Price **$14.99** (BIMQT)

Inside Macintosh®: QuickTime Components by Apple Computer, Inc.

Covers how to use and develop QuickTime components such as image compressors, movie controllers, sequence grabbers, and video digitizers.

List $34.95 Our Price **$17.50** (BIMQTCOM)

Inside Macintosh®: QuickDraw™ GX Programmer's Overview

- Provides an introduction to QuickDraw™ GX, providing an overview of the QuickDraw GX environment from a developer's perspective.
- Introduces the QuickDraw™ GX programming and runtime environments, the relationship between QuickDraw GX and the rest of the Macintosh® systems software and the relationship between QuickDraw GX and Macintosh applications.
- Learn the key elements of QuickDraw GX programming, data structures, object types, and functions used most frequently by QuickDraw GX developers are also covered.
- Provides a series of practical examples demonstrating how to approach programming with QuickDraw GX.

List $24.95 Our Price **$12.50** (BIMGXOV)

Inside Macintosh®: QuickDraw™ GX Graphics
by Apple Computer, Inc.

- Shows how to create and manipulate the fundamental geometric shapes of QuickDraw GX to generate a vast range of graphic entities.
- Demonstrates how to work with bitmaps and pictures, and specialized QuickDraw GX graphic shapes.

List $26.95 Our Price **$13.50** (BIMGXGR)

Inside Macintosh®: QuickDraw™ GX Objects
by Apple Computer, Inc.

Introduces QuickDraw GX and its object structure, and shows programmers how to manipulate objects in all types of programs.

List $26.95 Our Price **$13.50** (BIMGXOBJ)

Inside Macintosh®: QuickDraw™ GX Printing by Apple Computer, Inc.

- Essential for any developer whose QuickDraw™ GX application supports printing.
- Shows how to support the new printing features of QuickDraw GX, including desktop printers and expandable printing dialog boxes.
- Shows how to use printing-related objects to add custom panels to printing dialog boxes and to create custom page formats.

List $26.95 Our Price **$13.50** (BIMGXPRNT)

Inside Macintosh®: QuickDraw™ GX Printing Extensions and Drivers by Apple Computer, Inc.

- Essential for developers who want to create extensions to the application printing capabilities of QuickDraw™ GX, or who need to write a printing device driver that works with QuickDraw GX.
- Describes how to create printing extensions and printer drivers, and provides a complete reference to the messages, functions, and resources that they use.

List $29.95 Our Price **$14.99** (BIMGXEXT)

Inside Macintosh®: QuickDraw™ GX Typography by Apple Computer, Inc.

- Essential for developers who use QuickDraw™ GX to manipulate text.
- Shows how to use QuickDraw GX objects to handle all kinds of text – from plain, unstyled text to complex, mixed-direction and multi-language text with sophisticated stylistic and typographic variations.
- Shows how to create and manipulate the three different types of text shapes supported by QuickDraw GX including text shapes, glyph shapes, and layout shapes.

List $29.95 Our Price **$14.99** (BIMGXTYP)

Inside Macintosh®: QuickDraw™ GX Environment and Utilities by Apple Computer, Inc.

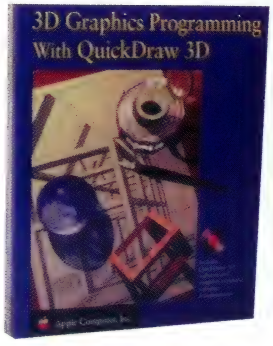
- Companion to QuickDraw™ GX Objects.
- Contains programming information useful to any developer writing QuickDraw GX applications.
- Describes QuickDraw GX memory management, error handling, debugging, and mathematical functions, as well as conversion from QuickDraw to QuickDraw GX.

List $29.95 Our Price **$14.99** (BIMGXENV)

3D Graphics Programming Using QuickDraw 3D
by Apple Computer, Inc.

- Incorporate spectacular 3D graphics into your applications.
- Explore QuickDraw 3D, a revolutionary graphics extension to the Mac OS for Power Macintoshes.
- CD contains the complete QuickDraw 3D system itself and a complete database of the QuickDraw 3D API, allowing you instant access to the hundreds of graphics calls via a fast viewing engine. Book/CD-ROM, 640 pages.

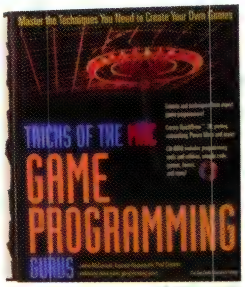
List $39.95 Our Price **$35.96** (B3DGRAP)



3D Game Machine v1.2 by Virtually Unlimited

- Create lightning-fast 3D arcade games and interactive multimedia applications.
- Ultra-fast rendering – 15 frames per second on a 14" monitor completely texture-mapped, with a PowerMac 6100/60
- Create full "virtual" 3D worlds with six degrees of freedom, free-form texture mapping, shading, material and light properties, convex/cave polygons with unlimited vertices, unlimited light sources, dynamic hidden surface removal, special graphic modes for fast full-screen animation, collision detection, explosion simulation, 3D data importing.
- Simple easy-to-use interface. Runs on all Macs! Works with CodeWarrior.

Our Price **$299.00** + license. (S3DGAME)



Tricks of The Mac Game Programming Gurus

- For beginning to expert game programmers
- Complete overview of all the necessary components of game programming on the Macintosh.
- Packed with valuable tools, utilities, sample code, CodeWarrior™ Lite and game demos.
- QuickDraw 3D and Power Mac optimization and inside info on how Glypha III was created.
- Hundreds of tried-and-true tricks, tips, and insider secrets from well-known Mac game programming experts

List $50.00 Our Price **$45.00** (BTRICK)

Black Art of Macintosh Game Programming:

by; Kevin Tieskoetter.

- Develop your own 3D games in C on the Mac.
- Includes CD with project files for both Symantec C and Code Warrior
- Create freeform texture-mapped games and polygon graphics
- Control dynamic source code-all compatible as native to the Power Mac
- Write directly to the screen, bypassing QuickDraw

List $39.99 Our price **$35.99** (BBLACK)

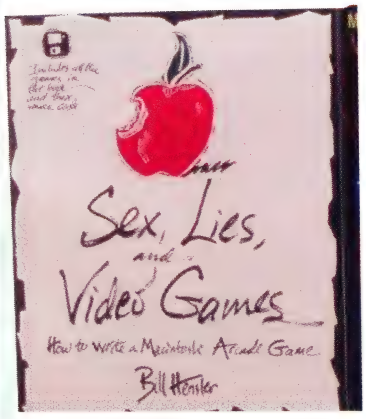
Includes CD-ROM!

Inside CodeWarrior 9

- Includes CodeWarrior IDE User's Guide.
- This is the printed version of the documentation provided on the CD.
- Covers CodeWarrior, the debugger, and associated tools.

Our Price **$34.95** (BINSCW)

SEE RELATED CATEGORY:
Dev. Environments



Sex, Lies and Video Games by Bill Hensler

Includes CD-ROM!

- A learn-by-example tutorial on the ins and outs of Mac arcade-style game programming in C.
- Features game theory, sprite animation, sound, and interaction techniques.
- A must-read for serious programmer's and hobbyists alike.

List 34.95 Our Price **$31.46** (BSEX)

SEE RELATED CATEGORY:
Tools, Libs & Utilities

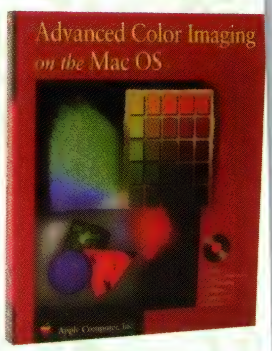
Graphic Gems V Edited by Alan W. Paeth

- Loaded with practical tools for implementing new ideas and techniques, to offer working solutions to real programming problems.
- Contains over 40 new gems in ellipses, splines, Bezier curves, and ray tracing – displaying the most recent and innovative techniques in graphics programming.
- Includes a disk with source code from all five volumes. Available in both IBM and Macintosh versions. CONTENTS: Algebra and Arithmetic. Computational Geometry. Modeling and Transformation. Curves and Surfaces. Ray Tracing and Radiosity. Halftoning and Image Processing. Utilities.

List $49.95 Our Price **$44.95** (BGEMS5)

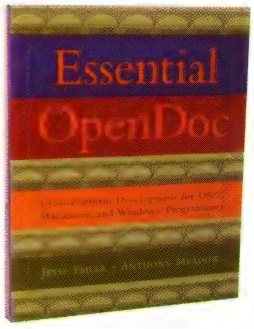
Includes Disk!

Advanced Color Imaging on the Mac OS



- Enhance your software's color capabilities with step-by-step instructions.
- Augment the color support supplied with QuickDraw, and QuickDraw GX.
- Use the Pallette Manager to get the best colors on limited displays.
- Match colors between screens and input/output devices (scanners & printers)
- CD includes a complete reference information in both QuickView and Acrobat formats. Plus, a sample application demonstrating ColorSync programming techniques.

List $36.95 Our Price **$33.25** (BADVCI)



OpenDoc Programmer's Cookbook

- Shows you how to create OpenDoc software components, called parts editors, for the Mac OS Platform.
- Including instructions for setting up the Macintosh Programmers Workshop (MPW) development environment to write OpenDoc software
- Annotated listings of explaining the methods that implement the SamplePart part editor
- Descriptions of other sample part editors created by the OpenDoc engineering team to illustrate more advanced features
- Summary descriptions of software utilities provided with OpenDoc for the Mac OS
- An Introduction to the System Object Model (SOM) technology underlying OpenDoc

List $24.95 Our price **$22.45** (BODCOOK)

Essential OpenDoc

- Gives an in-depth look at the technical issues of OpenDoc
- Explores the three core technologies that support it's functionality – SOM, OpenDoc's storage mechanism, and the Open Scripting Architecture (OSA).
- Also examines CyberDog, a set of OpenDoc part editors that provides access to Internet services and offers compelling example of the power of OpenDoc development

List $24.95 Our price **$22.46** (BESOD)

Inside CodeWarrior 9

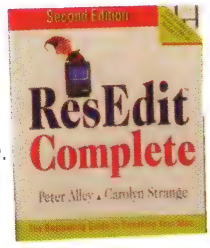
- Includes CodeWarrior IDE User's Guide.
- This is the printed version of the documentation provided on the CD.
- Covers CodeWarrior, the debugger, and associated tools.

Our Price **$34.95** (BINSCW)

SEE RELATED CATEGORY: **Dev. Environments**

ResEdit™ Complete, Second Edition

by Peter Alley and Carolyn Strange



- Customize every aspect of your interface form creating screen backgrounds and icons to customizing menus and dialog boxes. 608 pages. Book/disk package.

List $34.95 Our Price **$31.45** (BRESED2)

Inside PowerPlant Manual

- Create PowerPlant applications using the CodeWarrior IDE and PowerPlant Constructor.
- Full descriptions of major PowerPlant classes and resources.
- Included are the PowerPlant Constructor Manual, including View, TextTraits and Custom Types editing, and PowerPlant Library Reference, covering all classes and functions in PowerPlant.

Our Price **$34.95**

SEE RELATED CATEGORY: **Dev. Environments**

OpenDoc Programmer's Guide by Apple Computer, Inc.

- The official reference for the implementation of OpenDoc on the Mac OS.
- Describes the component software revolution and explains how to develop for it on the Mac OS platform.
- Accompanying CD-ROM contains a complete reference to the OpenDoc programming interface, and an extensive collection of tested, reusable sample code.

List $44.95 Our Price **$40.46** (BOPENDOC)

C++ Programming with CodeWarrior by Jan L. Harrington

- Beginning OOP for the Macintosh and Power Macintosh and Mac OS compatibles.
- Learn object-oriented programming techniques using C++ as the example language and Metrowerks and CodeWarrior as the example compiler.
- Enclosed CD contains example code from the book and a full-function Metrowerks CodeWarrior

List $35.95 Our Price **$32.35** (BCPPCW)

The ResEdit All Night Diner by David Ciskowski

- An idea-filled menu and introduction to the joys of customizing software.
- Add personality to the Mac by customizing default icons, the text of menus and dialog boxes, cursors, pointers and more.
- Disk features ResEdit, plus lots of sample resources

List $24.95 Our Price **$22.45** (BRESDINE)

Includes Disk!

SEE RELATED CATEGORY: **Dev. Environments**

C++ Programming W/MacApp by David Wilson, Larry Rosenstein & Dan Shafer

- Learn the secrets to unlocking the power of MacApp®, Apple's development environment for C++
- Learn to design complex windows and views using the ViewEdit tool
- Learn to support multipage text and graphics with only five lines of code
- Learn to support Undo for menu commands and drawing operations that use the mouse.

List $34.95 Our Price **$31.46** (BCPPMACAP)

SEE RELATED CATEGORY: **Tools, Libs & Utilities**

Programming in Symantec C++ for the Macintosh by Judy May and John Whittle

- An introduction to object-oriented programming, the C++ language, and Symantec C++ for the Macintosh.
- Great for both programmers and beginners alike.
- Covers everything from the basics to advanced features of Symantec C++.
- Includes helpful examples of C++ code that illustrate object-oriented programs.

List $29.95 Our Price **$26.95** (BPSYMCPP)

SEE RELATED CATEGORY: **Dev. Environments**

Symantec C++ Programming by Neil Rhodes & Julie McKeehan

Symantec C++ Programming for the Macintosh is a tutorial for getting up and running in the Symantec C++ environment, while mastering the techniques of object-oriented programming.

- Explore the Symantec C++ environment, from debugging a program and using resource utilities to building applications and creating objects
- Design programs for compatibility with multiple application frameworks
- Learn how to use the new Visual Architect, Inspector, and templates.

List $45.00 Our Price **$40.50** (BSYMCPP)

SEE RELATED CATEGORY: **Dev. Environments**

Metrowerks CodeWarrior Programming by Dan Parks Sydow

- Includes CodeWarrior Lite, and Full Coverage of PowerPlant™.
- The best information on Metrowerks CodeWarrior, giving full coverage to the Gold Edition.
- CD includes Code Warrior Lite.

List $39.95 Our Price **$35.95** (BCWPROG)

Includes CD-ROM!

SEE RELATED CATEGORY: **Dev. Environments**

Dan Shafer Presents the Power of Prograph CPX

- Master the revolutionary graphical object-oriented programming language.
- Step by step course through three interrelated projects of increasing complexity.
- Learn Prograph language, CPX classes and object editors.
- Includes disk with all code in the book.

List $49.95 Our Price **$19.95** (BDANPRO)

Order Toll-free
800-MACDEV-1
(800-622-3381)

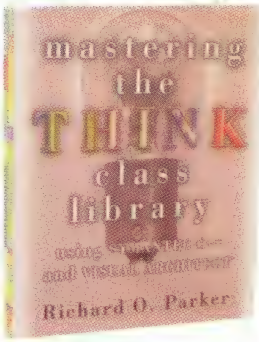
Visual Programming with Prograph CPX

by Scott B. Steinman and Kevin G. Carver

- An introduction to the language and a guide for advanced users, for both Macintosh and Windows-based machines.

List $34.00 Our Price **$30.60** (BVISPRO)

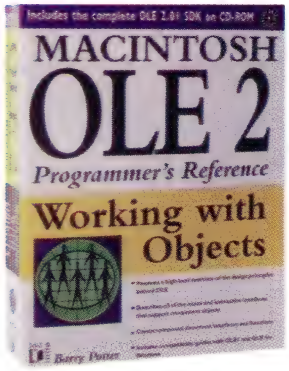
Mastering the THINK Class Library by Richard Parker



- Provides a thorough examination of Symantec's extensive Class Library and the Visual Architect.
- A complete description of the structure and operation of the TCL includes explanations of all code generated by the Visual Architect, any necessary custom code, and the operation of this code.
- Visual Architect tutorials provide you with a step-by-step approach for simplifying the development of complex Macintosh applications. 496 pages.

List $29.95 Our Price **$26.95** (BMASTERTCL)

SEE RELATED CATEGORY: **Dev. Environments**



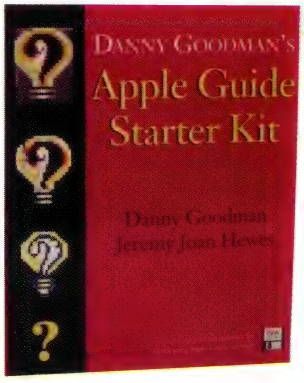
Macintosh OLE2 Programmer's Reference: Working with Objects

- Complete reference to the extensible protocol of Object Linking and Embedding, version 2.01 for Macintosh System 7.
- Describes the visual and interactice interfaces that support the component objects.
- Provides details of the OLE 2.01 for the Macintosh user Interface, addresses the issues of object class registration, shows how to implement the drag and drop objects from one application to another, covers the interface that exposes the basic embedding functionality, includes descriptions of API functions and more!

List $44.95 Our Price **$40.45** (BOLE2)

Danny Goodman's Apple Guide Starter Kit

by Danny Goodman and Jeremy Joan Hewes.



- Create your own Apple Guide databases quickly and easily, without having to learn a scripting language, write coded files, or use several different files and programs
- Includes advice and tips on how to design a good Guide, from planning and creation through testing, revising, and indexing. Book/disk, 320 pages.

List $34.95 Our Price **$31.46** (BDGAGSK)

MacsBug Reference & Debugging Guide For MacsBug version 6.2

by Apple Computer, Inc.

- MacsBug is an assembly-language-level debugging tool
- Macros, templates, dcmds, and other resources for making debugging easier
- Macintosh memory management and the operating system as they relate to low level debugging
- How to display and set memory and process registers
- Disassemble memory, and set execution breakpoints
- Discipline, a tool for testing the validity of toolbox parameters
- Debugging strategies you can use to find and cure common bugs
- Includes MacsBug 6.2.

List $34.95 Our Price **$31.46** (BBUGREF)

AppleGuide Complete

by Apple Computer, Inc.

- Covers Guide Maker, the software you use to build and test guide files.
- Learn about the complete cycle of designing as well as advanced topics such as scripting and coding guide files. Book/CD-ROM, 544 pages.

List $39.95 Our Price **$35.96** (BAPLGD)

SEE RELATED CATEGORY: **Tools, Libs & Utilities**

Programming For The Newton: Software Development using NewtonScript

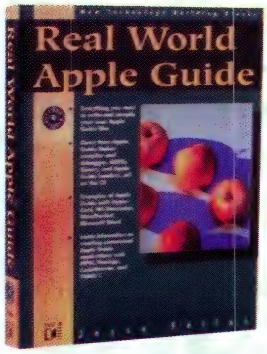
by Julie McKeehan and Neil Rhodes. Foreword by Walter R. Smith

- An indispensable tool for Newton programmers.
- Includes disk with sample Newton application from the books, as well as demonstration version of Newton Toolkit (NTK) – the complete development environment for the Newton®.
- A Publication of AP Professional May 1994, Paperback, 393 pp.

List $29.95 Our Price **$26.95** (BPROGNEWT)

SEE RELATED CATEGORY: **Dev. Environments**

Real World AppleGuide For The Mac

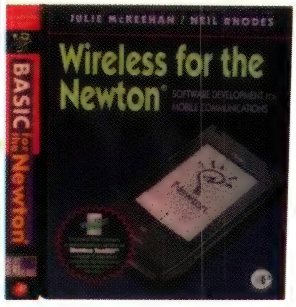


- An introduction to AppleGuide for programmers. It explains its design and function, plus how to design your own guides using AppleScript.
- Includes a disk of sample AppleGuides for AppleGuide-compliant applications.

Includes Disk!

List $39.95 Our Price **$35.95** (BREALWLD)

SEE RELATED CATEGORY: **Tools, Libs & Utilities**



Wireless For The Newton: Software Development for Mobile Communications

by Julie McKeehan and Neil Rhodes

- Learn to develop Newton® software on the Macintosh.
- Hands-on Newton environment training with sample code
- Includes disk with sample source code for a Newton application, as well as demonstration NTK™ – the complete development environment for the Newton®.

List $34.95 Our Price **$31.45** (BWIRELESS)

Programming for the Newton Using NS BASIC

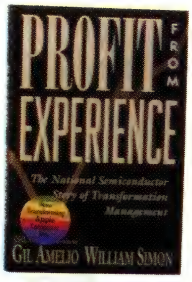
by John Schettino & Liz O'Hara

Includes Disk!

- Program on Macintosh, Windows-based PC, or on the Newton itself.
- Straight-forward "programming by example" approach – you'll be writing Newton programs right away.
- Includes 3.5" disk containing Demonstration NS BASIC and over fifty example programs. (Newton not included)

List $35.95 Our Price **$32.35** (BPROGNEWT)

SEE RELATED CATEGORY: **Dev. Environments**



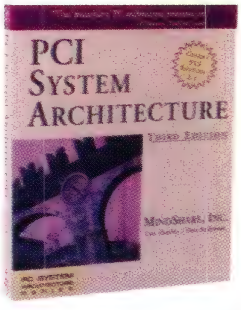
Profit From Experience by Gil Amelio and William Simon

Written by the new CEO of Apple

- The story of the transformation of National Semiconductor – how Amelio and his management team took it from it's worst loss in 30 years to the highest earnings in it's history • Includes: 6 core business issues and why they are critical to success • The TEAM program • 10 personal attributes to strive for • Amelio's guidelines for General Managers • Attributes to look for when hiring.

List $24.99 Our price **$22.45** (BPROFIT)

PCI System Architecture, Third Edition by MindShare



Describing revision 2.1 of the Peripheral Component Interconnect (PCI) bus specification, this book explores PCI's relationship to the rest of the system. It includes an in-depth treatment of PCI to PCI bridges, the PCI BIOS, the 66MHz PCI bus, and more. 592 pages. List $34.95 Our Price **$31.46** (BPCISYS)

PowerPC System Architecture

by MindShare Inc. and Tom Shanley

- Describes the hardware architecture of the PowerPC systems, providing clear, concise explanation of the PowerPC specification, the template upon which all PowerPC processors are designed.
- Includes the specs for both the 32 and 64 bit implementations including • supervisor privilege level facilities • logical memory addresses • I/O and memory mapped I/O • address translation for segments, pages, and blocks • virtual paging • interrupts.

List $34.95 Our Price **$31.46** (BPPCARCH)

Inside Macintosh®: AOCE Application Interfaces

by Apple Computer, Inc.

- Shows how your application can take advantage of the system software features provided by PowerTalk system software and the PowerShare collaboration servers.
- Add electronic mail capabilities to your application, write a messaging application or agent, store information in and retrieve information from PowerShare and other AOCE catalogs.
- Add catalog-browsing and find-in-catalog capabilities to your application, write templates that extend the Finder's ability to display information in PowerShare and other AOCE catalogs.
- Add digital signatures to files or to any portion of a document, and establish an authenticated messaging connection.

List $40.45 Our Price **$20.25** (BIMAOCE)

Inside Macintosh®: Interapplication Communication

by Apple Computer, Inc.

Shows how applications can work together. How your application can share data, request information or services, allow the user to automate tasks, communicate with remote databases.

List $34.95 Our Price **$17.50** (BIMIAPP)

Inside the Macintosh Communications Toolbox

by Apple Computer, Inc.

- The definitive reference to the Macintosh Communications Toolbox, an integral part of the System 7 Macintosh Toolbox that enables developers to create communications applications or add communications features to other applications.
- Describes all of the routines that provide programmers with standard access to important communications services and in addition enables programmers to extend the reach of the Macintosh into non-Apple environments.

List $24.95 Our Price **$12.50** (BCOMM)

Inside Macintosh®: Networking

by Apple Computer, Inc.

- Describes how to write software that uses AppleTalk networking protocol
- Describes the components and organization of AppleTalk and how to select an AppleTalk protocol.
- Provides the complete application interfaces to all AppleTalk protocols, including ATP (AppleTalk Transaction Protocol), DDP (Datagram Delivery Protocol), and ADSP (AppleTalk Data Stream Protocol), among others.

List $29.95 Our Price **$14.99** (BIMNET)

Inside Macintosh®: AOCE Service Access Modules by Apple Computer, Inc.

- Describes how to write a software module that gives users and PowerTalk-enabled applications access to a new or existing mail and messaging service or catalog service.
- Shows how to write a catalog service access module (CSAM), a messaging service access module (MSAM), and AOCE templates that allow a user to set up a CSAM or MSAM and add addresses to mail and messages.

List $26.95 Our Price **$13.50** (BIMAOCES)

Inside Macintosh®: Text by Apple Computer, Inc.

- Describes how to perform text handling, from simple character display to multi-language processing.
- Covers Font, Script, Text Services, and Dictionary Managers, in addition to QuickDraw Text, TextEdit, and International and Keyboard Resources.

List $39.95 Our Price **$19.99** (BIMTEXT)

Inside Macintosh®: Devices by Apple Computer, Inc.

- Describes how to write software that interacts with built-in and peripheral hardware devices.
- Learn how to write and install your own device drivers, desk accessories, and Chooser extensions.
- Communicate with device drivers using the Device Manager; access expansion cards using the Slot Manager; control SCSI devices using SCSI Manager 4.3 or the original SCSI Manager.
- Communicate directly with Apple Desktop Bus devices; interact with the Power Manager in battery-powered Macintosh computers, and communicate with serial devices using the Serial Driver.

List $29.95 Our Price **$14.99** (BIMDEV)

Inside Macintosh®: X-Ref by Apple Computer, Inc.

Is a fast access to all the information in Inside Macintosh. Inside Macintosh X Ref; Provides programmers with a quick and easy way to find the exact information they need in this definitive suite of books, (all 26 volumes). It is indexed by topic, volume, chapter, and accompanying page number.

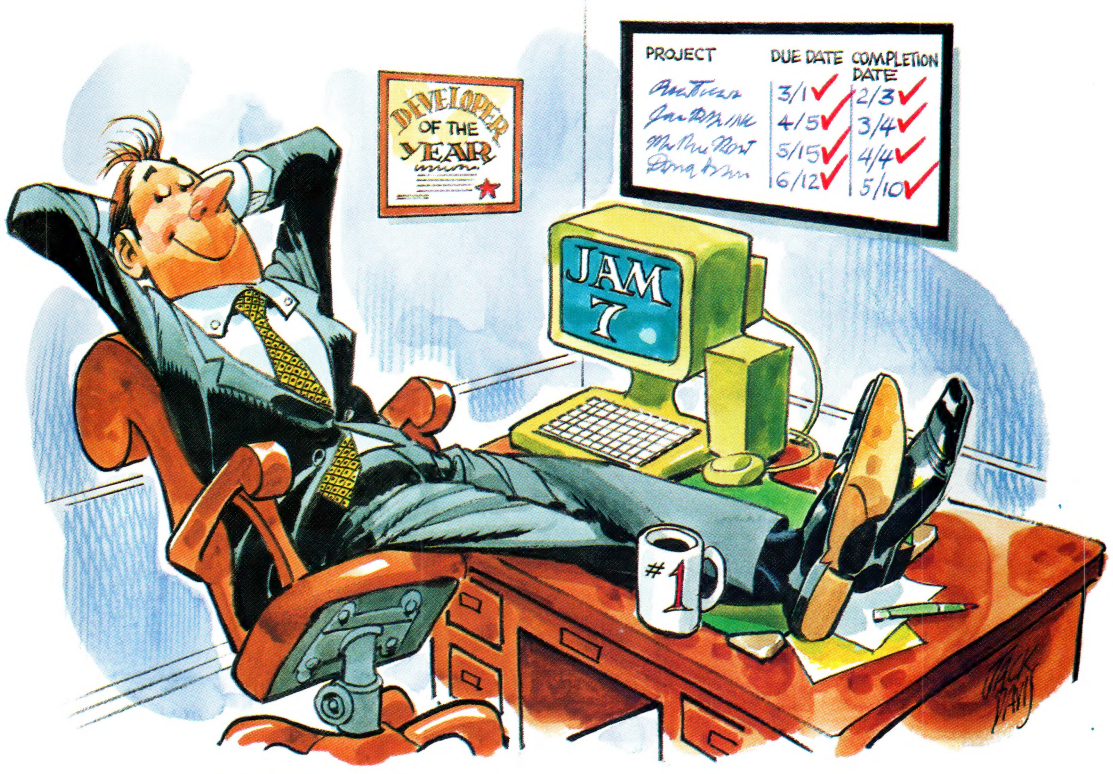
List $19.95 Our Price **$9.99** (BIMXREF)

Order Toll-free
800-MACDEV-1
(800-622-3381)

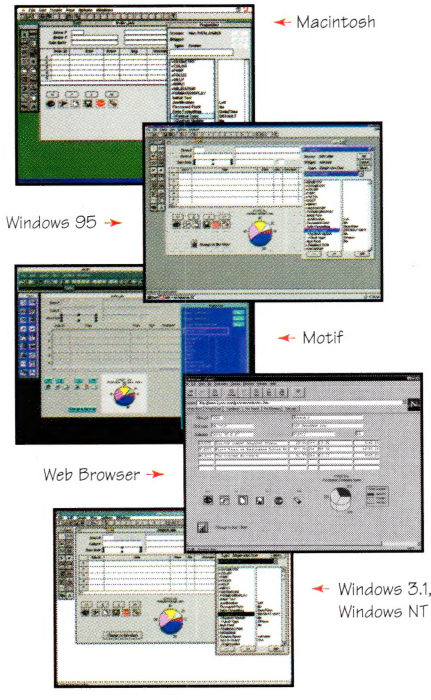
All entries in this index are alphabetized.
For your convenience the product names are **bold**, book names are *italicized* and company names are in plain text.

INDEX

Life is Sweet with JAM!



Get JAM and enjoy success in building and deploying client/server and Web applications



In a world where requirements change and deadlines don't, only JAM gives you what you need to build powerful, completely portable client/server, 3-tier and Web applications. JAM's cross-platform RAD capabilities enable you to successfully develop and deploy demanding applications across any platform, database, GUI or Web Browser.

Are you willing to base your project's success on a tool that only promises portability? JAM lets you build great looking applications that are fully portable today. If you need to build serious client/server, 3-tier or Web applications, call JYACC for a free demonstration kit or white paper.

- ✓ Desktop ease with the power of a 2nd-generation tool
- ✓ Unrivaled portability between Windows® 3.1, NT, 95; Macintosh®; OS/2 Warp®; Character, Motif and Web Browsers
- ✓ Unparalleled database access to Oracle®, Sybase®, Informix®, ODBC and more
- ✓ Automatic SQL generation with full transaction control
- ✓ Visual repository-driven development of both client and application server
- ✓ Centralized control over application objects via inheritance
- ✓ Unique architecture for team development
- ✓ No runtimes

Find out for yourself how sweet life can be with JAM!

Call 1 800 458-3313.

Or E-mail: sweetlife3@jyacc.com for a free demonstration kit or white paper.
Visit our Web site at http://www.jyacc.com
For international inquiries call: 1-212-267-7722 or FAX 1-212-608-6753

JYACC
Meeting the needs of professional developers for over 18 years

BRAZIL (55) 11 816 6228 • FINLAND (358) 3486 4000 • FRANCE (33) 1 46 92 45 44 • GERMANY (49) 40 79 70 07 0 • HRVATSKA (385) 1 242 116 • ISRAEL (972) 3 557 3675
ITALY (39) 2 45 2701 • MEXICO (52) 5 663 0405 • RUSSIA (7) 095 288 1924 • SAUDI ARABIA (966) 2 665 8406 • SINGAPORE (65) 220 8322 • SLOVENIA (386) 61 1405 004
SPAIN (34) 1 804 0625 • SWEDEN (46) 8 15 10 10 • SWITZERLAND (41) 21 991 9041 • THAILAND (66) 2 513 3559 • THE NETHERLANDS (31) 70 320 9214 • UNITED KINGDOM (44) 171 814 6660